Utrecht University

Master of Science,
Artificial Intelligence,
Intelligent Agents

Course Planner

Group 8

Francesco Pontiggia

Harmen Heida

Markos Polos

Shu Zhao

Thomas Fink

31-10-2019

Prof. Pinar Yolum

Contents

1. Project Introduction

The goal of the project described in this report was to design and develop an agent that will help a new master student to choose courses. The agent is going to suggest a schedule for the student based on the preferences and obligations of the student, for example when he/she is able to take the course or which course or topics likes. The agents' Knowledge Base is stored in a Web Ontology Language (OWL) file, edited and analysed using Protége. A goal-based agent has been chosen as an agent structure. The Java programming language is used to query the ontology and develop the agents' program. The ontology is queried with the HermiT reasoner.

2. Ontology

An ontology is a description of the world representation of an agent. The structure of concepts and relations of the ontology presented in this report ontology can be seen below in image 1 and image 2.

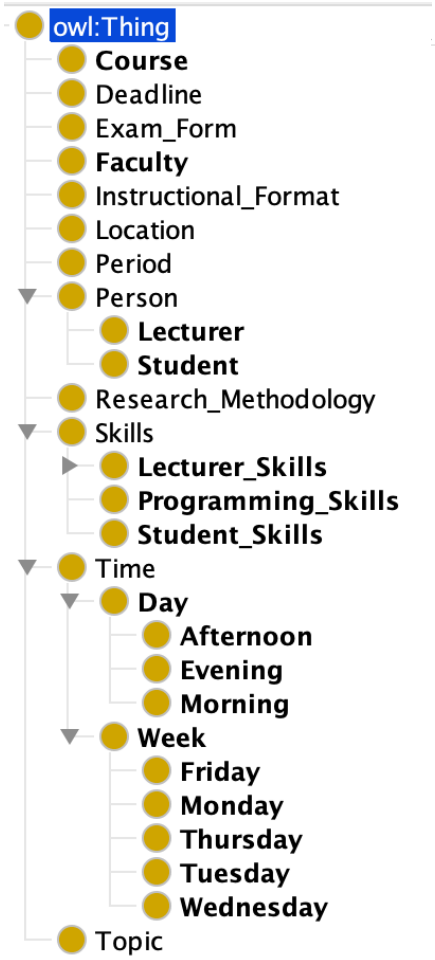

Image 1: Ontology class structure

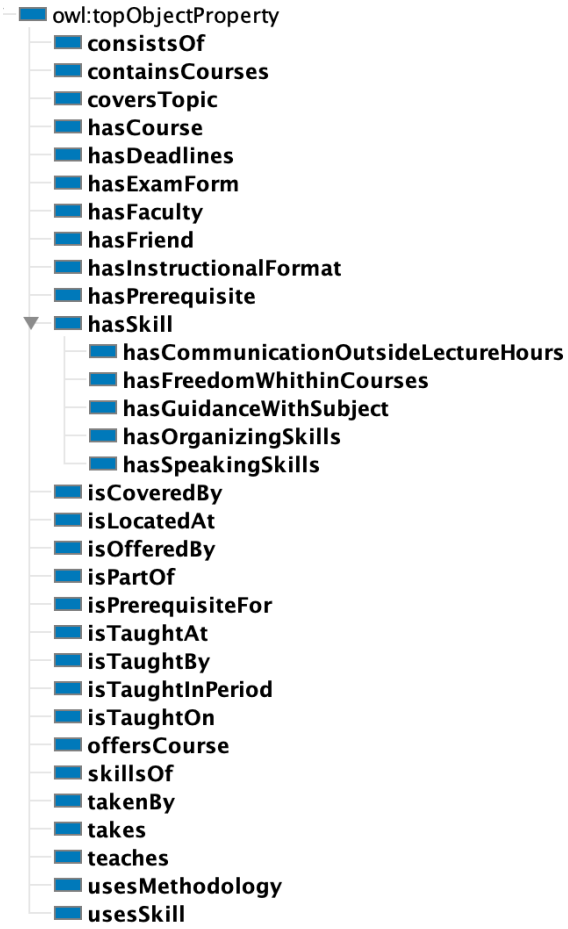

Image 2: Relations

2.1 Concepts

The ontology is divided into several concepts that are going to be explained in the following section of the report. Omitted from further explanation will be the concepts given in the project description. The before mentioned concepts are *Course*, *Topic*, *Research_Methodology*, *Lecturer* and *Student*.

In the ontology, you will find a couple of small concepts that fulfil the purpose of defining the Course concept in a more specific way. This will give the students more possibilities when defining their preferences for a whole year schedule. As these concepts define a pretty similar function within the ontology they will be explained as a whole. The before mentioned concepts are Deadline, Exam_Form and Instuctional_Format where Deadline is the frequency of deadlines within a Course (Ex. weekly), Exam_form is the form of given exam (Ex. Presentation or Written exam), Instructional_Format is the format a course is given in (Ex. Lecture or Lab).

It could be said that Deadline, Exam_Form and Instructional_Format broaden the specifics of a course as they are part of the Course subclass, where Course is a minimum of 1 Exam_Form a minimum of 1 Instructional_Format and exactly one Research_Methode as these concepts do not make an inference to other concepts and nothing is inferred from them. As each concept instance is disjoint from the other concept instances within the corresponding concept the following statements can be made. Course hasConcept "value" or Course not hasConcept hasConcept value where concept in hasConcept can be any of the above-specified concepts and value an instance of the said concept.
An example of deadline is shown below in image 3.

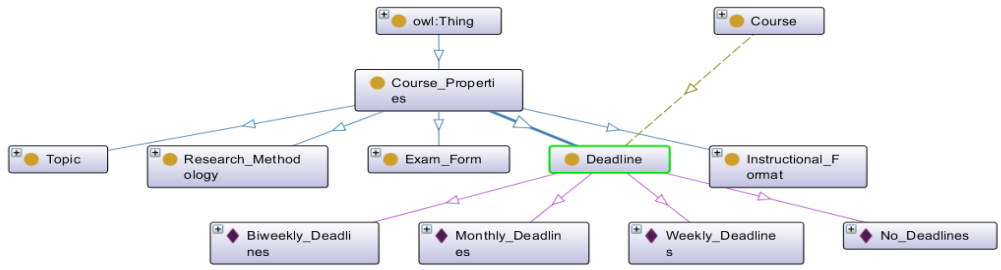

Image 2: Course_Properties class

The ontology also implements the Faculty concept. This concept defines all the faculties a Course can be offered by (Ex. beta science). Faculty is strongly connected with the location concept as a faculty can only have one location and one location can only have one faculty. With these concepts combined the user gets the opportunity to define preferences where a location is preferred or omitted in the results. The following queries can be made to achieve the previous stated. Course isTaughtAt "value" or Course not isTaughtAt "value". In the image below a direct relation between course and location is not shown, this is because it uses a special relation which is explained more extensively within the Relations section of the report.

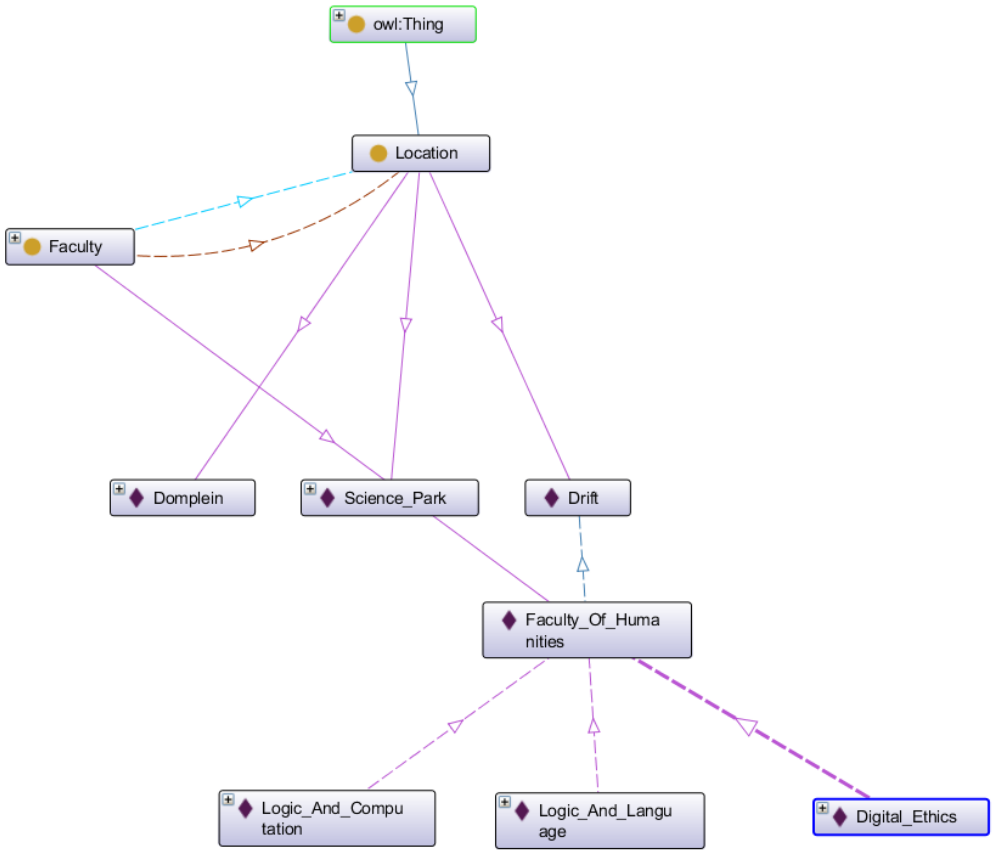

Image 3: Location

In the course description is defined that a course is a concept that is taught on exactly two different day-concept. Hence, the ontology should contain a notion of time. As seen in the image below the concepts of time are defined as follows. Week, all the days of the week (Ex. Monday) as a subclass of the week concept and the parts of a day (Ex. Monday_Afternoon) as a subclass of the day concept. The reason behind this implementation is that a course is a subclass of exactly two Week concepts which implies that a course must consist of two day part instances. In the ontology the day concepts are disjoint with each other, hence a course can only be a thing that is taught on exactly two different days.
Because the notion of time for a course is in dayparts the user can be pretty specific on which time the courses should or shouldn't be which the ontology can infer because the day-part instances are disjoint per day concept.

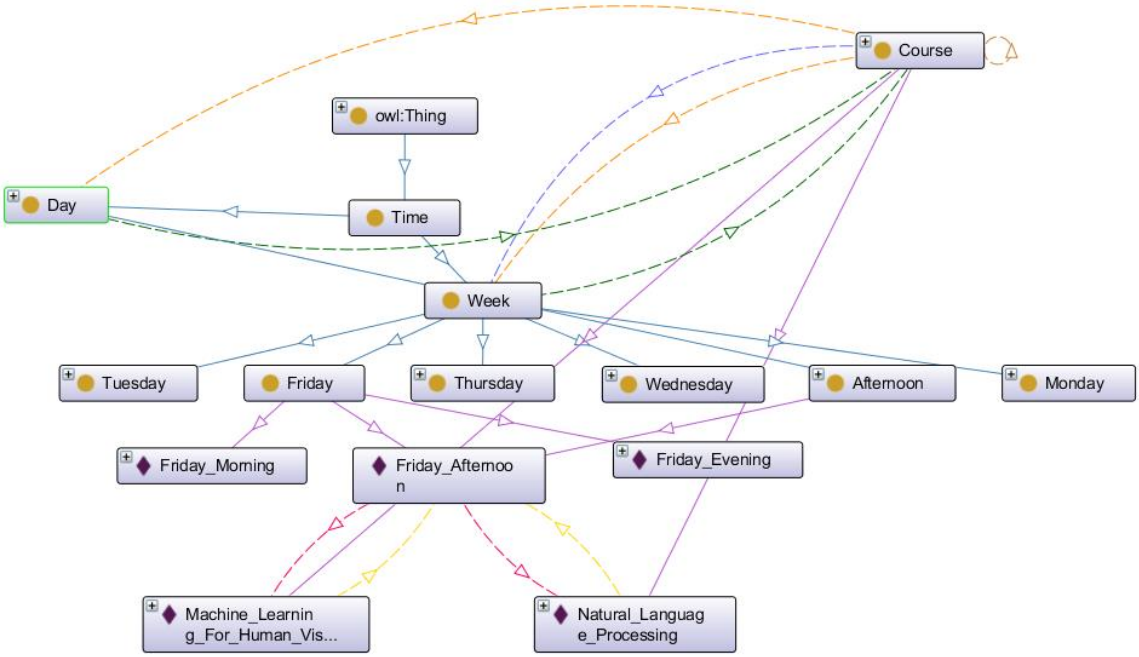

Image 4: Time class

Too expand on lecturer the concept skill was added to the ontology. This concept captured the skills a lecturer could have (Ex. Low_Speaking). This was a student could define preferences on the characteristics of lecturer. The skills of a lecturer have been divided into concepts that are a subclass of the skill concept as can be seen in the image below. This way a user could give a preference like hasSpeakingSkill "High_speaking". Because every instance of a skill concept was disjoint with all the other instances in its corresponding concept the following could be specified not hasSpeakingSkill "Low_Speaking". The former would yield the result of lecutars that have every speaking skill except Low_Speaking.
The skill concept covers more than lecturer skill as it also holds the skill a course would require or a skill that a student would have, for example programming. By this you could infer if a student has the appropriate skills to complete a course.

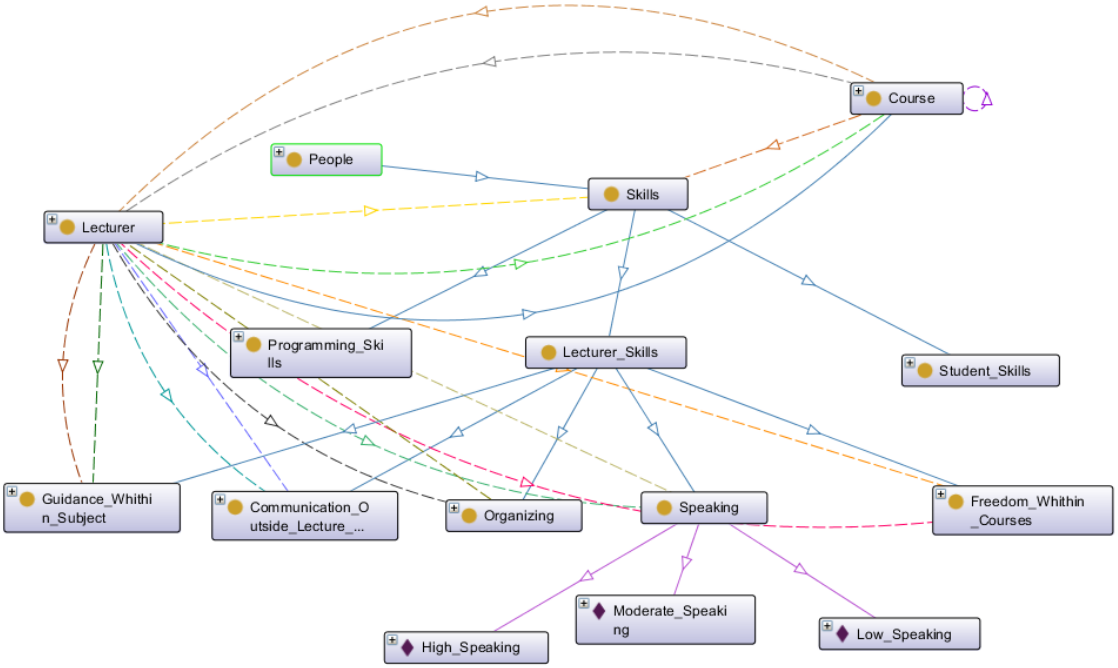

Image 5: Skills class

Relations

The ontology describes relations between different concept instances. The most interesting relations are described in this section of the report.

The functional relations of the ontology are HasDeadlines, all the subrelations of *HasSkill*, *isLocatedAt*, *isOfferedBy*, *isTaughtinPeriod*. Their semantics is respectively that a course has some Deadlines (which could be monthly, biweekly, weekly, no Deadlines), a lecturer has some specific quality level per each skill (low, moderate, high), a Faculty is located at some specific place, a course is offered by a faculty and a course is taught on a specific period. They are functional due to their semantics.

The only symmetric relation is *HasFriend*, which connects all the students already in the Ontology which are friends with each other.

Interestingly enough, *hasPrerequisite* is not a transitive relation, since there are no chains of prerequisites between courses at UU so this kind of inference would be useless.

Topics are related to each other via the relations *consistOf* and *isPartOf*. The semantics of these relations is as follows: a topic can be divided into (*ConsistsOf*) one or more (sub)topics, and can compose (*isPartOf*) one or more (super)topics. The *consistOf* relation is then the inverse relation of *isPartOf*. All the topics are in this way organized into a hierarchy, which could be quite intricate, since every node (topic) can have more than one parent (supertopic), or child (subtopic), creating a structure which cannot even be represented by a tree. Moreover, both the relations

are obviously transitive, due to their semantics. The reasoner has the task of inferring the inverse relation from the *isPartOf* one, and creating the whole structure so that with simple queries it is possible to find all the topics related to other topics. For example, if a user wants to get a course on the topic *Business_Topic*, which is a very general topic (and then has a high position in the hierarchy), but also something more specific into that field, this requirements could be stated simply with *coversTopic some (isPartOf value Business_Topic)*; on the other hand, if he wants to get a course which covers at least something about the specific topic *Advanced_HCI_Qualitative_Research_Methods_Topic*, it could be stated simply with *coversTopic some (consistsOf value Advanced_HCI_Qualitative_Research_Methods_Topic)*.

Another interesting relation is *IsTaughAt*, which quite straightforwardly specifies that a course is taught at a specific location. It is completely inferred by the reasoner, since it's simply defined as the concatenation of the relation *IsOfferedBy* (which combines a course with the faculty offering it) and *isLocatedAt* (which combines a location with the respective location in Utrecht city). As an example, if intelligent Agents is offered but the Faculty of Science and the Faculty of Science is located at the Science Park, then the reasoner is able to infer that Intelligent Agents is taught at the Science Park.

3. Goal-based agent

Agent program

The agent senses user preferences which are generated in the user interface. The user is not able to express how much he/she weighs each preference. The task of the agent was designed as generating a full schedule given these unweighted preferences. Unfortunately, it is not a realistic scenario to generate a complete schedule while taking into account all preferences. This is since each preference describes a smaller concept than the original Course concept in the ontology, which itself consists of a small number of course-instances.

Hence, it was decided that the agent should have an action to drop some preferences. Since the preferences are unweighted the agent cannot assume which preferences the user prefers to drop. Therefore it was decided that the agent should try to drop the least amount of preferences.

So, the task of the agent can be described even better by: the agent should generate a complete study path, while taking into account the most user preferences possible. This a clear goal, which can be easily transformed to a computer-interpretable rule. Therefore it has been chosen to develop a goal-based agent.

However this is a clear goal, the goal is dynamic as well. "Finding a study path, while respecting all user preferences" is the most strict goal the agent handles. The goal "Finding a study path, while respecting the most user preferences" clearly is a less strict goal. The agent presented in this report is able to dynamically change its goal to a less strict goal. An algorithm which handles dynamic goals is presented in the next paragraph.

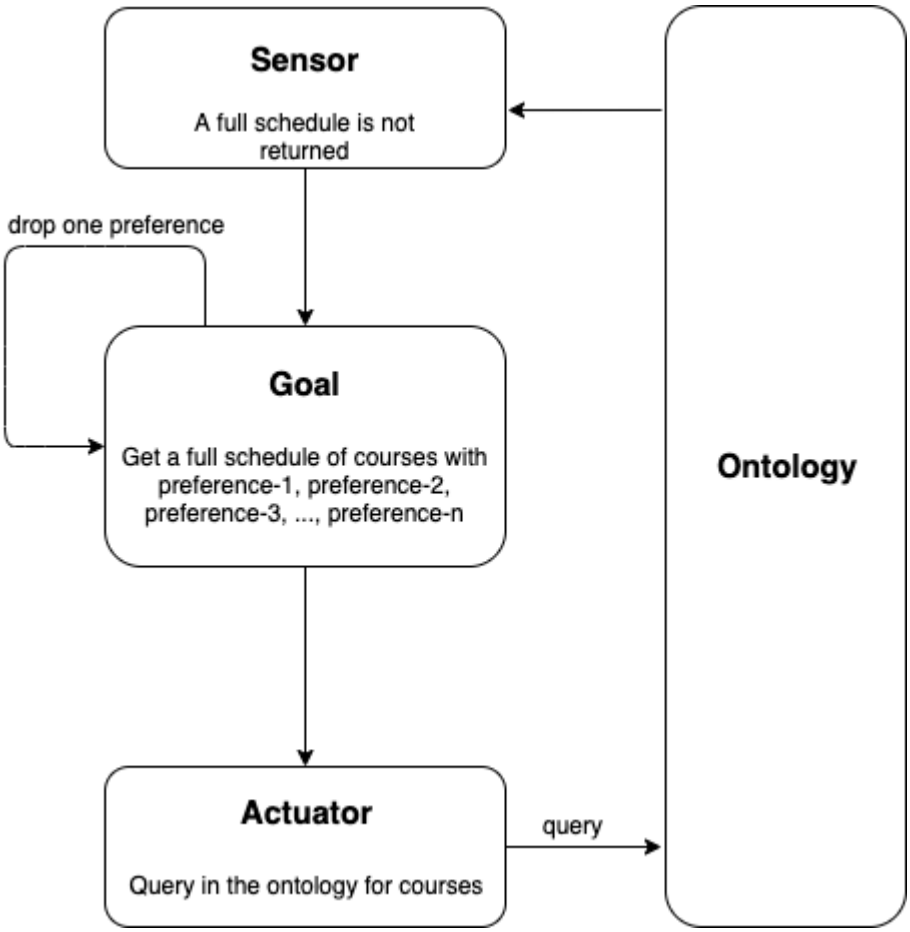

Image 6: Goal based agent architecture

To structurally decrease the strictness of a goal, breadth-first search(BFS) algorithm has been implemented. The goal check method of the BFS algorithm checks the current state(=node) by sending the corresponding query to the ontology. If the ontology returns two or more courses, the goal is met. If the goal is not met, the BFS will generate all possible succeeding states and place them on a queue until a state is found in which the goal is reached.

The method in the BFS algorithm that returns the current state its child nodes, generates these nodes by performing the agent actuators on the current state. The actuator of this agent deletes one single preference of the query to generate less strict queries. The method splits the query on the 'and' operator and generates all possible children nodes in which only one preference is deleted. So the preference query shrinks with every layer of nodes in the BFS tree. An example is shown in the image below.

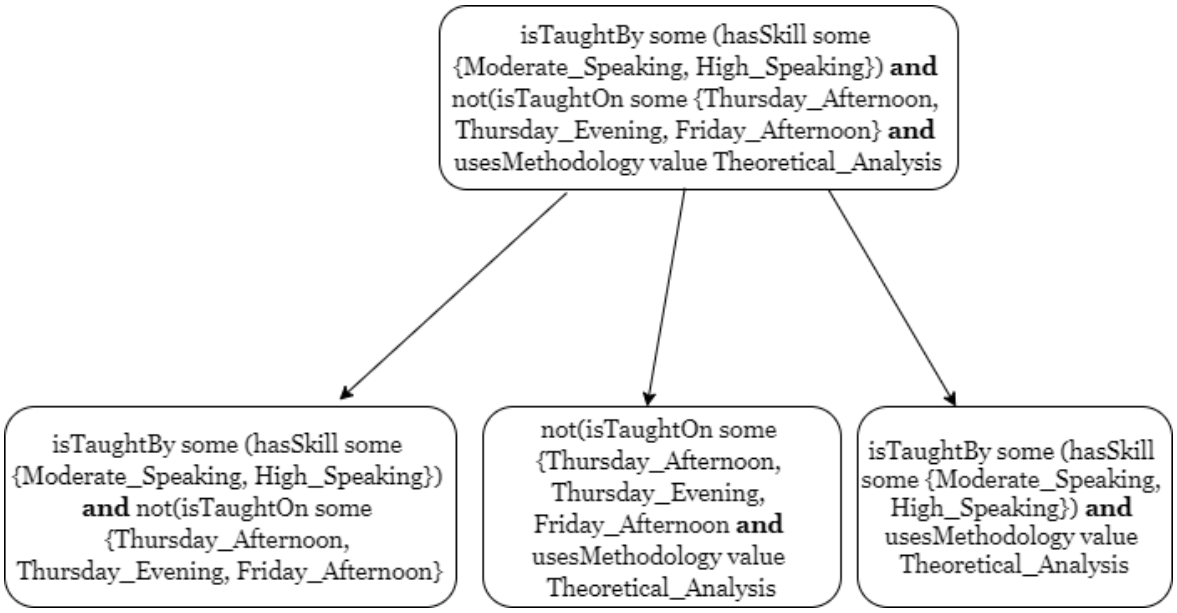

Image 7: BFS graph example

Deleting single preferences from the current query string is done in a structured manner. That is, every non-static preference is deleted once in the set of children nodes. The static preference is the isTaughtInPeriod relation. These queries can never be deleted, since they are specific for the search that is performed. For example, by deleting the isTaughtInPeriod relation the ontology will not only return courses in period 1 but also courses in the other periods.

Due to the properties of the BFS, if the agent finds a goal, it is always the shortest action-sequence to the goal. Moreover, the agent will always find a goal, since at the end every preference in a query can be dropped. Then only the static preferences are in the query; isTaughtInPeriod will give more than two results for every period.

Thus, the agent uses a BFS algorithm to find an action sequence by which the goal can be reached. Moreover, the agent is aware of its goal, since a threshold is present in the goal-check function within the BFS. Due to the properties of the ontology and BFS, a goal will always be found, and no shorter complete action sequence exists. Hence, the course planner agent is a goal-based agent.

Conflicts

When a goal is found the BFS algorithm will always first check for time-based-conflicts. That is, two courses cannot be taught to a student at the same part of a day. If any conflict occurs, the agent checks first if more than two courses are found in the goal state. If so, the agent checks if one of the extra courses does not have a conflict with one of the other courses, such that a two-courses per period program can be made. If not, the agent returns to the BFS and dequeues the next state for which a goal-check will be invoked. If this state consists of enough courses the agent will again check for conflicts until a conflict-free goal state is found.

If this conflict-free goal state is found, the agent found the definite action sequence to reach a goal. In the BFS the actions are already executed by the method that generates new nodes/states, as mentioned above. Hence, the agent can start with a new achievement, which can the next period or helping a new user.

Multi-agent course planner

The agent is able to ask for the evaluation of a course from other online agents, and agent itself will keep track of the beta reputation scores of these neighbouring agents, thus this agent will only adopt the evaluation from the agent with the highest beta reputation score. The agent registry and discovery mechanism is based on Netflix Eureka framework, and the communication between agents is over HTTP.

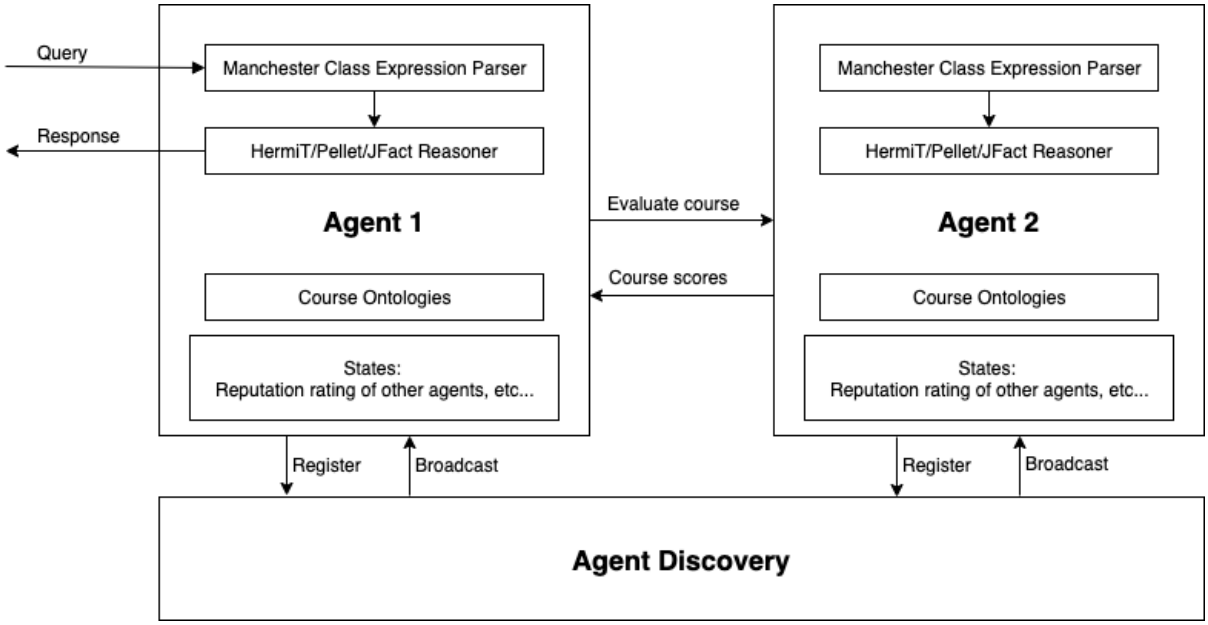

Image 8: Agent Discovery

Agent architecture

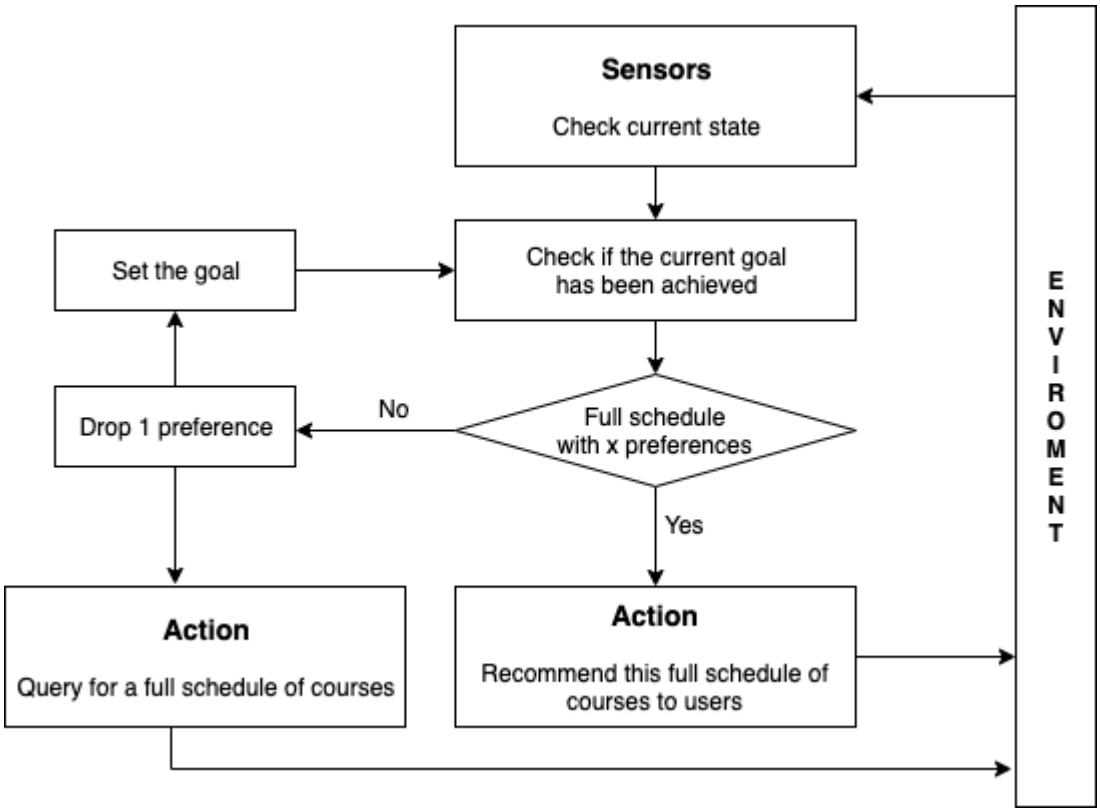

Image 9: Agent architecture

In this part the agent architecture will be described through the above-illustrated diagram of the agent.

First, as stated in the agent program section, the **goal** of the agent is to recommend to the user a full schedule of courses with respect to the user's preferences. The agent will keep checking whether the goal has been achieved or not, and if it detects that the current goal is not able to be achieved with the current set of preferences from the user, it will drop one preference and reset the goal. This will be done by the algorithm described above, which is implemented in Java code.

Second, the **state** consists of agent state and world state, which is able to be perceived by the sensors from the environment:

World state	**S1**: Online neighbouring agents
	S2: Course evaluation from neighbouring agents
Agent state	**S3**: Total registered courses
	S4: Available time slots in the calendar

	S5: Currently-kept preferences from the user
	S6: Reputation rating of neighbouring agents

The **action** stands for the action that the agent is able to take, which will influence the internal agent states and the environment by the actuators:

Action
A1: Recommend a full schedule of courses to the user
A2: Query for a full schedule of courses
A3: Reset the goal
A4: Ask for the evaluation of a course from neighbouring agents
A5: Answer the evaluation of a course from neighbouring agents
A6: Give positive or negative feedback for evaluations of courses from neighbouring agents

How the world evolves (which is perceived by the sensors) and what my actions do (which is actuated by the actuators) will both have an effect on the world and agent states:

Sensors	Actuators	State
E1: Neighbouring agents online		**S1**: Online neighbouring agents
	A4: Ask for the evaluation of a course from neighbouring agents **A5**: Answer the evaluation of a course from neighbouring agents	**S2**: Course evaluation from neighbouring agents

	A2: Query for a full schedule of courses	
	A3: Reset the goal	**S5**: Currently-kept preferences from the user
	A1: Recommend a full schedule of courses to the user	**S3**: Total registered courses **S4**: Available time slots in the calendar
	A6: Give positive or negative feedback for evaluations of courses from neighbouring agents	**S6**: Reputation rating of neighbouring agents

Lastly, the **knowledge base** is built on the ontologies, which contains the following consistent facts the courses, which will be queried by the agent and inference will be drawn based on the chosen reasoner: HermiT, Pellet or Fact++.

Facts about a course	Lecturers
	Faculties
	Research Methodologies
	Topics
	Time slots
	Location

4. Scenarios

To evaluate the agents performance on both the ontology and the search-algorithm five fictive scenarios have been made up, which can be found in the Appendix A. In these scenarios a student is described whom is starting its master Artificial Intelligence at the Utrecht University. The students have diverse preferences about the properties of the curriculum. For each scenario the preferences are converted to a manchester syntax query, which can be seen under the scenario. Since preferences can differ between periods, queries exist for each period. The scenarios have been designed in a way they consist of multiple preferences such that the search algorithm can be tested on dropping preferences in a query.

The Manchester syntax queries include the intersection(‘and’) and complement(‘not’) operators. The union operator is not used in the scenarios, since

the agents' search algorithm will only split the intersection operator. Moreover, using 'or' between preferences A and B has the same meaning as using the 'and' between A and B, since the BFS algorithm will explore the following combinations: both A and B, only A, only B and none. Furthermore, the existential restriction('some') is used, sometimes referring to a set that only consist of one element ('value').

The queries will both consist of parts for which no inference is needed. The query "isTaughtInPeriod value Period_1", which is part of every scenario, is more a database lookup because all this instance relations are entered in the ontology. On the other hand, a part of the first period query of scenario 1 "isTaughtBy some (hasSkill some {Moderate_Speaking, High_Speaking})" needs some inference of the reasoner. First the reasoner has to find the instances of the concept that is described by "hasSkill some {Moderate_Speaking, High_Speaking}", such that this concept can function as the range of the isTaughtBy relation.

The queries that have included the complement operator 'not' are the most difficult to infer. For the reasoner to infer the some instances do not have an object property P, the definitions of this property P have to be very strict. A good example of such a query is "Course and not(isTaughtOn some {Monday_Afternoon, Tuesday_Morning})", which can be found, inter alia, in scenario 4. This sentence describes the concept of Courses which are not taught on Monday afternoons and Tuesday mornings. To infer this concept, first all the part-of-day instances have been made disjoint. Moreover courses are defined by being taught on exactly 2 parts of day instances.

5. Performance measurement

The performances are divided into functional and not functional: on one side the functional performances regarding the achievement of the goal (how far is the agent from the achievement of the goal?), on the other the non functional performances regarding how the goal should be achieved (did it do it in a good way?). For both of them some evaluation metrics are presented here. For a non functional metrics, some requirements on the results can be stated, while functional metrics can work properly only to compare different agents.

Functional metrics

To evaluate the results of the queries, and to allow some kinds of comparisons between different results of different agents, the following performance metrics have been developed. Although the usual metrics of a confusion matrix are presented here (precision, accuracy and recall) it's still not possible to build a confusion matrix out of them, since it's not a prediction problem and it's not possible to define the usual categories (TP, FP, TN, FN). For example, an incorrect plan which has been refused by the user would be a TN, but it is not a desirable situation, whereas it is desirable in the prediction problem of a machine learning algorithm. Anyway, they are a useful and already known method of expressing and evaluating results.

- **Precision of generated plans**: A plan presented by the system is defined as correct if it satisfies all the requisites of the student. A plan is defined as partially correct if it satisfies all the requirements except for at most 3 of them. A plan is defined as incorrect if it is not even partially correct. Precision is defined as follows:

Precision = $(Corrects + 1/2\,Partials) \div (Corrects + Partials + Incorrects)$.
Of course this value is highly influenced by the number of requirements of the user. That's why there is the next metric.

- **Recall:** if the system declares that a query is not consistent, it still presents a plan with the minimum number of not respected requisites. Than the plan can be accepted by the user, or refused. Recall is defined as follows:
  Recall = $Accepted \div (Accepted + Refused)$
- **Total Accuracy:**
  Accuracy = $(Corrects + 1/2\,Partials + Accepted) \div (Corrects + Partials + Incorrects + Accepted + Refused)$
- **Number of respected requisites**
- **Number of not respected requisite**

Since these values depend on the behaviour of the user, it's recommended to use them only for comparison between different agents.

Non functional Metrics

- **Time of response**: the time the system needs to elaborate a plan and load the result web page. An acceptable value should be below 2 seconds. A desirable value should be below 0,5 seconds.
- **Expressivity**: The same system must be usable by different users without changing the code neither the ontology to allow for new specific requisites of the user. This implies not only that the user interface should be able to let the user express all his preferences, but also, with greater relevance for the purpose of this project, that the ontology should contain all the necessary relations, data properties and concepts.

Evaluation

All the assessment part has been strongly limited by the time all the queries have required to obtain a result: at least five minutes are required to execute each query. It can be stated that the time of response requirement hasn't been respected. For more information, see the Discussion part. On the other hand, the expressivity requirement has been respected, for the given scenarios.

6. Discussion

In this report a goal-based agent has been presented which makes use of an ontology in a study curriculum domain. In the report the ontology and the agents' structure is discussed. The agent is tested on fictive scenarios, which are written for the same domain as the ontology. In this last part of the report, some hurdles and future works will be discussed.

A negative aspect of this agent is the amount of time it consumes to come up with results. The agent performs a BFS for every of all four periods. For every node the algorithm has to query the ontology for a goal check. The BFS only stops when a goal has been found. In the worst case the BFS will go on till its first leaf node, where no preferences exist. Hence, with larger queries the complexity is increasing

tremendously. In addition, the agent has to checks for conflicts every time a possible goal is found. This in combination, costs the agent between five and ten minutes to complete one task. This time consumption is not realistic for an actual usage of this agent in real life. Improvements have to be made to shorten the time the agent has to reason about its actions. These improvements can be found in the ontology, cleverly integrating the ontology into the agent, the agents program and in the agents structure.

Making an ontology that will make use of the reasoners' capabilities have been found difficult. Inference is limited to: inverse functions, some negations of object properties, the hierarchy of topics and the concatenation of relations. Some of the queries sent to the ontology are sort of a database lookup, since no inference has to be made. For example "isTaughtBy value Pinar_Yolum" is available in the ontology without inference. More use of the ontology has to be made to make an efficient and more rational agent.

For example, time slot-conflicts are checked by the code part of the agent itself. However, an ontology has the capability to infer such properties as well. Then, instead of checking for conflicts every time a state has more than 2 courses, the ontology will never return conflicting courses because it reasoned so.

The agent described in this report does not add any new knowledge to the ontology. The agent is using the ontology more or less as a database to get information about the concepts described in it. By first adding new, user knowledge, to the ontology, the reasoner could infer new information. The queries followed by the agent would give back more accurate results, which makes it much quicker to get the right results.

More investment should be made to develop a knowledge base in the form of an ontology. The knowledge base is the basis of the agent. By reasoning, the agent could be able to infer new knowledge. Next, some more additions to the ontology are described, which will probably make more inferences possible.

Concatenation of functions, as well as inverse functions, is a powerful way of exploiting the reasoner capabilities. While the second ones are widely used in the agents' ontology, the first one is used only one time: for future improvements of the ontology, using it more times is imperative.

A student class, with corresponding definitions, would allow the reasoner to perform a lot of new inferences. A simple design of it could be as follows. Add a simple relation Prefers, with subrelations PrefersTimeslot,PrefersTopics and so on; then add a simple relation Hates, with subreleations HatesTimeslot, HatesTopic and so on; when a new user inserts his preferences, a new instance of the Class relation is created with all these relations to have them in the ontology. The subrelations can be splitted again to allow to express preferences per each period, but can always be not period specific to express for example the requirements "I do not want to take any course in the topic Artificial Intelligence". Add a relation PrefersTaking which collects all the courses which matches the Prefers… relations. Add a relation HatesTaking which collects all the courses which matches the Hates…relations. Add a relation Takes which is initially filled with all the mandatory courses. When a user asks for recommendations, the ontology returns all the courses of the PrefersTaking relation (maybe with an inverse relation inferred by the ontology, IsPreferredBy value Student_Name).

This leaves open the opportunity of an interaction with the user: if the user does not like the result (maybe because the requirements were too strict and there aren't enough results) then he can change them and ask again for the result, in a kind of loop. The student should also be allowed to fill the Takes relation by registering to a Course. Defining properly the Student class would allow to avoid the enrolment in more than 3 courses per period. This would involve a kind of exception handling, and it's the only part of the design whose feasibility is still in doubt. A relation HasCompleteSchedule would allow to know which students have chosen at least 2 courses per period, and then their schedule is acceptable, and which students haven't done yet (this could be useful for the system administrator e.g.).
At least the following definitions for Course would be added to specify some requirements on the schedule:

- The definition "takes max 1 (isTaughtOn value Monday_Morning) and takes max 1 (isTaughtOn value Monday_Afternoon) and … " would define that a student cannot take more than two courses on every individual part of day.
- the definition "takes max 3 (isTaughtOnPeriod value Period_1) and takes takes max 3 (isTaughtOnPeriod value Period_2) and …." would allow to specify that a student can't take more than 3 courses per period.

The search algorithm uses (Breadth-First Search) is time and memory consuming, since many states are being stored and considered. Other, more greedy, algorithms do not have this disadvantage. These algorithms will find a goal. However, it is not guaranteed that this goal is the most optimal solution, which respects the most possible preference kept. This disadvantage could be solved by for example a depth limited search. If the agent (or the user) accepts, for example, a study path in which 70% of the preferences are respected, the depth limit should be set to the ceiling of the number of preferences * 0.3. For a query consisting of 10 parts, the search will delete three times one preference of the query before making another combination. Most likely, more improvements in time and memory consumption could be made by experimenting with different search algorithm.

In this paper the agent describes is a goal-based agent which uses unweighted preferences. In the future, weighted preferences could be used, which would make it meaningful to make utility-based agent. Being able to implement an utility-based agent has some advantages over a goal-based agent. As discussed in this report most preference queries are to strict to return any courses from ontology inference. Therefore parts of the preference query has to be dropped. A preference weight, set by the user, could help to decide what preferences to drop first such that the preferences most important to the user are kept in the final study path. This agent could be easily incorporate weighted preference by implementing best first search instead of breadth first search. Best First Search will first try the states with the least weighted amount of preferences dropped. Another way to implement weighted preferences to this agent is by designing a utility function. We propose the following utility function:

$$utility = 0.6 \times courseUtility + 0.4 \times preferenceUtility$$

$$courseUtility = \frac{Max(0, amountOfCourses^2) + (4 \times amountOfCourses)}{4}$$

$$preferenceUtility = \frac{amountOfPreferences}{totalPreferenceAmount}$$

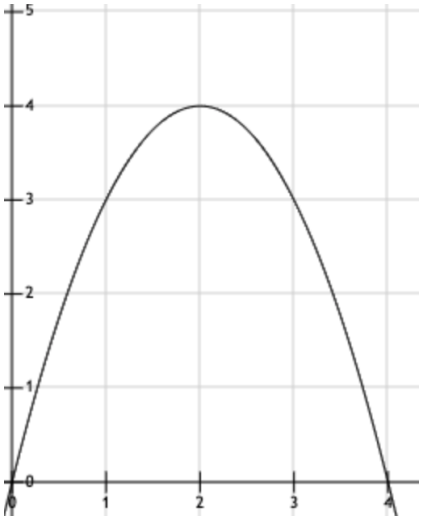

Image 10: Course amount utility function

With the function stated above the utility can be calculated. Said utility function gets a utility based on the amount of courses found and on the preferences satisfied in the result. The function for the course amount will have a utility of zero when there are zero courses found and a utility of one when two courses are found and zero again when four courses are found as shown in the graph above.
The preference utility is a simple function defined by taking the amount of preferences satisfied divided by the total amount of preferences given by the user. These two utilities are combined into one total utility where it values the course amount utility for 60% and the preference utility for 40%. The reason for this distribution is that the correct amount of courses is valued greater than matching more preferences. For example if you have one course found with a lot of preferences satisfied the schedule would still be incomplete.

The availability of new potential algorithms, which goes together with using a utility function, could potentially improve the agents' computational speed. Because of the utility function, the search space could be expressed as a graph with local and global optima. Several algorithms that search for a global optimum have been studied for other use cases. Based on the available literature a search algorithm could be chosen that serves a search problem in the study-curriculum-domain well.

Another negative aspect of this agent that if a conflict free list of at least two courses is being found the first three courses in this set are proposed. This is a random/alphabetic approach of choosing courses for the user. Better would be to incorporate a utility function for this choosing point as well. Courses could be ranked on there utility, the above stated function can be used to calculate this ranking. Then

the first two or three courses could be picked for the final study path. We propose the following course utility function which can be seen above.

6. Appendix

A. Scenarios

1.

"Josè Carrillo is an international student who has just arrived at UU for his master degree and he is looking for courses for year 1. Since he is not used to the language yet, he wants to take courses only from lecturers with at least moderate talking skills, to be sure to properly understand all the lectures.

Besides this, he wants to take a course from H.M. Huistra, since she went to Josè's bachelor university for a conference and Josè appreciated it so much to decide to come to Utrecht for his master.

To pay for his studies Josè is going to work during the winter: he has already found a part-time job in the afternoon on Wednesdays and Thursdays. Due to that, his courses must not be taught in period 2 and period 3.
Josè has understood that in the summer the Wilhelmina-park is full with students enjoying the sun with beers and wines. Therefore he only wants to follow courses on Thursday and Friday mornings, later these days he is going to the park with his friends. On Monday, Tuesday and Wednesday he is available the whole day

His goal is to do a PHD in Logics, so he is mainly interested in a theoretical approach to research, in order to explore all the existing literature and gain a wide knowledge of the fundamental milestones of every field. Therefore he wants to take some courses that cover Logic Topics. Moreover, he doesn't want to follow any philosophy courses, since he is bad at it.

He is a very theoretical person: he prefers doing written exams, where he takes better grades with respect to other forms of evaluation.
"

<u>Period 1-search</u>

isTaughtInPeriod value Period_1 **and** isTaughtBy some (hasSkill some {Moderate_Speaking, High_Speaking}) **and** isTaughtBy value H_M_Huistra **and** usesMethodology value Theoretical_Analysis **and** hasExamForm value Written_Exam **and** coversTopic some(isPartOf value Logic_Topic) **and** not (coversTopic some (isPartOf value Philosophy_Topic))

<u>Period 2-search</u>
isTaughtInPeriod value Period_2 **and** isTaughtBy some (hasSkill some {Moderate_Speaking, High_Speaking}) **and** isTaughtBy value H_M_Huistra **and** usesMethodology value Theoretical_Analysis **and** hasExamForm value Written_Exam **and** not(isTaughtOn some {Wednesday_Afternoon, Wednesday_Evening, Thursday_Afternoon, Thursday_Evening}) **and** coversTopic some(isPartOf value Logic_Topic) **and** not (coversTopic some (isPartOf value Philosophy_Topic))

Period 3-search
isTaughtInPeriod value Period_3 **and** isTaughtBy some (hasSkill some {Moderate_Speaking, High_Speaking}) **and** isTaughtBy value H_M_Huistra **and** usesMethodology value Theoretical_Analysis **and** hasExamForm value Written_Exam **and** not(isTaughtOn some {Wednesday_Afternoon, Wednesday_Evening, Thursday_Afternoon, Thursday_Evening}) **and** coversTopic some(isPartOf value Logic_Topic) **and** not (coversTopic some (isPartOf value Philosophy_Topic))

Period 4-search
isTaughtInPeriod value Period_4 **and** isTaughtBy some (hasSkill some {Moderate_Speaking, High_Speaking}) **and** isTaughtBy value H_M_Huistra **and** usesMethodology value Theoretical_Analysis **and** hasExamForm value Written_Exam **and** not(isTaughtOn some {Thursday_Afternoon, Thursday_Evening, Friday_Afternoon, Friday_Evening}) **and** coversTopic some(isPartOf value Logic_Topic) **and** not (coversTopic some (isPartOf value Philosophy_Topic))

2
"Since she graduated from Utrecht Hogeschool, Wendy Jansen has developed a deeply practical approach to the field of AI. Consequently, she wants to do at least two projects and but she hates to present these with a presentation. Considering that the previous 15 ECTS she has already taken in period 1 have involved no projects. She knows the lecturer Greco and definitely wants to take a course from him before graduating. Because of her interest in the practical approach of AI she wants to follow courses about AI within games and no courses about a Computation Topic

She lives in the countryside near Utrecht: she prefers to avoid lectures early in the morning due to striking farmers. Moreover, she is going to miss a lot of classes due to the distance: in her case, it's necessary that the lecturers are available for advice and help even outside lecture hours.

Before graduating, she wants to experience studying at the Utrecht Centre buildings, since she likes the atmosphere in de centre.
".

Period_1-search
isTaughtInPeriod value Period_1 **and** hasExamForm value Project **and** not (hasExamForm value Presentation) **and** isTaughtBy value G_Greco and not (isTaughtOn some Morning) **and** isOfferedBy some (isLocatedAt some {Drift, Domplein}) **and** hasSkill value High_Communication_Outside_Lecture_Hours **and** hasSkill value High_Guidance_Whithin_Subject **and** coversTopic some (isPartOf value AI_For_Game_Technology_Topic) **and** not (coversTopic some(isPartOf value Computation_Topic))

Period_2-search
isTaughtInPeriod value Period_2 **and** hasExamForm value Project **and** not (hasExamForm value Presentation) **and** isTaughtBy value G_Greco and not (isTaughtOn some Morning) **and** isOfferedBy some (isLocatedAt some {Drift, Domplein}) **and** hasSkill value **and** coversTopic some (isPartOf value AI_For_Game_Technology_Topic) **and** not (coversTopic

some(isPartOf value Computation_Topic)) High_Communication_Outside_Lecture_Hours **and** hasSkill value High_Guidance_Whithin_Subject

Period_3-search
isTaughtInPeriod value Period_3 **and** hasExamForm value Project **and** not (hasExamForm value Presentation) **and** isTaughtBy value G_Greco and not (isTaughtOn some Morning) **and** isOfferedBy some (isLocatedAt some {Drift, Domplein}) **and** hasSkill value High_Communication_Outside_Lecture_Hours **and** hasSkill value High_Guidance_Whithin_Subject **and** coversTopic some (isPartOf value AI_For_Game_Technology_Topic) **and** not (coversTopic some(isPartOf value Computation_Topic))

Period_4-search
isTaughtInPeriod value Period_4 **and** hasExamForm value Project **and** not (hasExamForm value Presentation) **and** isTaughtBy value G_Greco and not (isTaughtOn some Morning) **and** isOfferedBy some (isLocatedAt some {Drift, Domplein}) **and** hasSkill value High_Communication_Outside_Lecture_Hours **and** hasSkill value High_Guidance_Whithin_Subject **and** coversTopic some (isPartOf value AI_For_Game_Technology_Topic) **and** not (coversTopic some(isPartOf value Computation_Topic))

3
"Linda Arden is a student who has already taken Intelligent Agents and Philosophy of AI and really appreciated those courses. Due to that, she is now deeply interested in the topics of Agents and Artificial Intelligence, and she would like to take courses related to those topics. Moreover, she liked the way those courses were taught (by professors Pinar Yolum, so she is looking for courses offered by a professor with the same characteristics.)

Josè Carrillo and Linda Arden form a good team for projects. She heard that in Period 2 there is going to be a big and useful project, however, doesn't remember the name of the course. She only remembers that is was part of the machine learning topic. Therefore she is looking for courses with projects as exam form in period 2. She wants to take this course with Josè.
"

Period_1-search
isTaughtInPeriod value Period_1 **and** not {Intelligent_Agents, Philosophy_Of_AI} **and** isTaughtBy some (hasSkill some (skillsOf value Pinar_Yolum)) **and** coversTopic some(isPartOf value Agents_Topic) **and** coversTopic some(isPartOf value Artificial_Intelligence_Topic)

Period_2-search
isTaughtInPeriod value Period_2 **and** not {Intelligent_Agents, Philosophy_Of_AI} **and** isTaughtBy some (hasSkill some (skillsOf value Pinar_Yolum)) **and** hasExamForm value Project **and** takenBy value *Josè_Carrillo* ***and*** coversTopic some (isPartOf value Machine_Learning_Topic) **and** coversTopic some(isPartOf value Agents_Topic) **and** coversTopic some(isPartOf value Artificial_Intelligence_Topic)

Period_3-search
isTaughtInPeriod value Period_3 **and** not {Intelligent_Agents, Philosophy_Of_AI} **and** isTaughtBy some (hasSkill some (skillsOf value Pinar_Yolum)) **and** coversTopic some(isPartOf value Agents_Topic) **and** coversTopic some(isPartOf value Artificial_Intelligence_Topic)

Period_4-search
isTaughtInPeriod value Period_4 **and** not {Intelligent_Agents, Philosophy_Of_AI} **and** isTaughtBy some (hasSkill some (skillsOf value Pinar_Yolum)) **and** coversTopic some(isPartOf value Agents_Topic) **and** coversTopic some(isPartOf value Artificial_Intelligence_Topic)

4
"Antoine Dumond, after graduating in Philosophy, has decided to enrol in the AI master degree at UU. He hasn't got a background in computer science, so he has poor skills in programming and mathematics. Due to that, he does not want to have courses that require some kind of knowledge in programming or maths in the first two periods. Antoine indeed is taking some additional courses to recover from this lacks, so he is already busy on Monday afternoon and Tuesday morning"

He prefers having a lot of free time and the opportunity to organize the activities on his own, to participate in as many as possible events and associations. Due to that, he is looking for lecturers who conceive high freedom to the students. Moreover, he is a friend of Linda Arden, so he would like (if possible) to follow the same courses as her. His favourite lecturers are R. Iemhoff and Dong Nguyen"

Period_1-search
isTaughtInPeriod value Period_1 **and** not(usesSkill some Programming **and** usesSkill some Math) **and** not(isTaughtOn some {Monday_Afternoon, Tuesday_Morning}) **and** isTaughtBy some {R_Iemhoff, Dong_Nguyen} **and** isTaughtBy some (hasSkill value High_Freedom_Whithin_Courses) **and** takenBy value Linda_Arden

Period_2-search
isTaughtInPeriod value Period_2 **and** not(usesSkill some Programming **and** usesSkill some Math) **and** not(isTaughtOn some {Monday_Afternoon, Tuesday_Morning}) **and** isTaughtBy some {R_Iemhoff, Dong_Nguyen} **and** isTaughtBy some (hasSkill value High_Freedom_Whithin_Courses) **and** takenBy value Linda_Arden

Period_3-search
isTaughtInPeriod value Period_3 **and** isTaughtBy some {R_Iemhoff, Dong_Nguyen} **and** isTaughtBy some (hasSkill value High_Freedom_Whithin_Courses) **and** takenBy value Linda_Arden

Period_4-search
isTaughtInPeriod value Period_4 **and** isTaughtBy some {R_Iemhoff, Dong_Nguyen} and isTaughtBy some (hasSkill value High_Freedom_Whithin_Courses) **and** takenBy value Linda_Arden

5

"Jan Jansen has already done during his bachelor degree a project in Python. Therefore, he wants to get acquainted even more with this language, by attending at least one course which involves programming in Python. Jan Jansen has never been in touch with ethics and philosophy. This year he wants to focus to get more into these topics.

Since he has a form of disability, he experiences some problems to move from a location to another one, so that all the courses should be in the Science Park. During the first two periods he is doing some kind of physiotherapy on Friday morning to recover from his problem, so he can't attend courses on that day. Nonetheless, since the lab is not accessible for disabled people, unfortunately, he is forced to discard courses (only in the first semester) which include lab classes

Moreover, he is attending a public speaking course at Parnassos Cultural Centre. Due to that, he can't attend classes on Thursday afternoon, and would like to follow a course which trains the presentation skills of the students, to test the efficacy of his hard-working"

Period_1-search
isTaughtInPeriod value Period_1 **and** isOfferedBy some (isLocatedAt value Science_Park) **and** not (isTaughtOn some {Friday_Morning, Thursday_Afternoon}) **and** hasExamForm value Presentation **and** not (hasInstructionalFormat value Lab_Format) **and** usesSkill value Python **and** coversTopic some(isPartOf value Philosophy_Topic)

Period_2-search
isTaughtInPeriod value Period_2 **and** isOfferedBy some (isLocatedAt value Science_Park) **and** not (isTaughtOn value Thursday_Afternoon) **and** hasExamForm value Presentation **and** usesSkill value Python **and** coversTopic some(isPartOf value Philosophy_Topic)

Period_3-search
isTaughtInPeriod value Period_3 **and** isOfferedBy some (isLocatedAt value Science_Park) **and** not (isTaughtOn value Thursday_Afternoon) **and** hasExamForm value Presentation **and** usesSkill value Python **and** coversTopic some(isPartOf value Philosophy_Topic)

Period_4-search
isTaughtInPeriod value Period_4 **and** isOfferedBy some (isLocatedAt value Science_Park) **and** not (isTaughtOn value Thursday_Afternoon) **and** hasExamForm value Presentation **and** usesSkill value Python **and** coversTopic some(isPartOf value Philosophy_Topic)