OntRAT: Ontology-based rules acquisition tool

A. BinSubaih[*,1]**, S.Maddock**[1]**,** and **D.Romano**[1]

[1] Department of Computer Science, University of Sheffield, UK

Virtual training environments require a flexible scenario creation system that can accommodate different training scenarios to address a wide range of situations for different levels of trainee expertise. Hence, the scenario logic or behaviour should not be hard-coded in the simulation engine and should be accessible to the trainer or domain expert. We present a rules acquisition tool called OntRAT that combines an ontology that represents and stores knowledge in a structured format and remains at a high-level of abstraction which is accessible by domain experts, and a set of rules that enables reasoning about the knowledge to solve problems. We demonstrate the use of OntRAT for a particular virtual environment we have built, showing how it facilitates knowledge representation that is accessible by domain-experts, end-users, and developers.

Keywords knowledge acquisition; knowledge representation; rule-based systems; ontologies

1. Introduction

Previous work in virtual environments, and especially training environments, has created systems that rely on hard-coded logic in the software or simulation engine (see [1] for a review). Our aim is to separate the logic from the environment it is being used in (which follows the recent development trend to use ‘business rules’ in developing software [2]). The usual cycle for creating the logic involves the knowledge engineer extracting knowledge from the domain-expert and passing it to the developer. This lengthy process is prone to ambiguities. Our approach is to give the domain-expert direct access to and control of a structured representation of the knowledge, which is also interpretable by a system and readable by a developer.

We have developed a tool called OntRAT that provides rules acquisition based on an ontology [3], and we have used this as part of an architecture we have developed for virtual environments [1] that we are using to train police officers how to deal with traffic accidents. Figure 1 gives an overview of the architecture. Our tool gives the trainer direct management over two parts of a virtual environment: the content and the logic. The content (entities, attributes and constraints) is represented as an ontology and a set of rules (in the format of ‘if condition then action’) represents the logic.

For our traffic accident domain, example entities are: drivers, passengers, witnesses, officers, vehicles, skid-marks, broken glass, road, etc. Some of these entities (i.e. drivers, passengers, witnesses, and officers) share similar constraints (e.g. name, address, age, position, alcohol-level, etc) and in ontological engineering they are grouped together by a ‘concept’. In this case, a suitable concept is ‘person’. Concepts can be further structured in a hierarchical format similar to classes in object-oriented design. An example rule in our domain is: “if the driver’s alcohol-level is beyond x then inform the officer to apprehend him”. Using our ontology this can be represented as: “if driver.alcohol-level > x then officer.hint=‘Apprehend’ + driver”.

There are other tools that can be used to create ontologies, e.g. Protégé-2000 [4], OilEd[1], and PC Pack 4[2]. Also, languages such as OWL have started to appear to formally represent ontologies. However, ontologies do not include rules that represent actions. Tools such as VisiRule[3], and EZ-Xpert 3.0[4] can be used to create and manage rules. However, these do not use ontologies. There appears to be no tool that

[*] Corresponding author: e-mail: a.binsubaih@dcs.shef.ac.uk, Phone: +44 1142221877
[1] http://oiled.man.ac.uk/
[2] http://www.epistemics.co.uk/
[3] http://www.lpa.co.uk/vsr_det.htm
[4] http://www.ez-xpert.com/

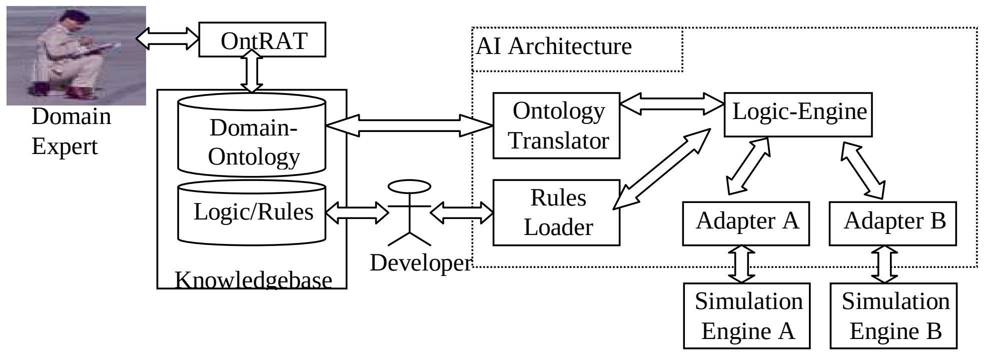


Fig. 1 System overview showing how OntRAT is linked to the AI Architecture

combines the two representations. The closest is JessTab [5], which is a bridge, not an acquisition tool, between rules formatted in Jess rule patterns and ontologies represented in Protégé-2000.

Using OntRAT, a domain-expert can store knowledge using the 'ontology and rules' representation which our architecture then makes use of, e.g. to provide trainees with hints on how to complete tasks. The current architecture automatically translates the ontology representation to the reasoning-engine representation. However, the rules are currently manually translated to the reasoning engine format by a knowledge engineer or programmer, as illustrated in Figure 1. The domain-ontology and rules databases are serviced to different simulation engines via adapters, and the simulation engines are the visual interfaces through which end-users interact with virtual environments.

Section 2 will discuss knowledge representation, section 3 will present the knowledge creation process, showing screenshots of OntRAT, Section 4 will present brief results of using the knowledge in a virtual training scenario, and section 5 will present conclusions.

2. Knowledge Representation

Knowledge representation is paramount to making the tool accessible to all parties (domain-experts, developers, and end-users) involved in the system. We had to choose a representation that adheres to two primary requirements: structure with high-level abstraction, and ability to reason with the knowledge to solve problems. The high-level abstraction objective is to represent the knowledge to domain-experts and end-users in a meaningful way. The structure, however, aims to ease the developers' task by providing them with knowledge that is structured thus enabling it to be interpreted and manipulated by the system. The second requirement goal is for the knowledge to hold information that is capable of reasoning with knowledge to solve problems in a way that closely resembles how experts would solve problems.

We found no single representation that could satisfy both requirements. This forced us to look at combining multiple representations to satisfy the requirements. We examined logic-based representation, associated network representation, frame-based representation, object-oriented representation, ontology representation, and rule-based representation. Detailed description of these representations can be found at [3,6].

Logic-based representation is very expressive but it lacks structure and the ability to represent logic that does not evaluate to true or false. The associated network, on the other hand, allows for better structure than logic-based representation in the form of arcs between nodes. Its deficiencies lie in the non-existence of interpretation standards and the difficulty in maintaining the network for complex knowledge. The frame-based and object-oriented representations are better structured than the previous two representations however they are considered as low-level representations, not to be exposed to domain-experts or end-users. Also, the fact that they allow procedural representation in the form of methods makes it difficult to consider them as a high-level abstraction. The ontology representation is easily understood and maintainable; combining these features with the success domain-experts have had in building libraries of ontologies across different disciplines (such as biology, health, eLearning, e-business, etc) and the existence of many tools which use an ontology as the base representation (e.g. Protégé, OilEd, OntoBuilder, OntoEdit, Ontoloingua, etc) makes an ontology the ideal representation to satisfy our first requirement.

Although some of the previous representations are capable of satisfying the second requirement of conducting problem solving through the use of procedural programming we wanted something that resembles the way experts solve problems without imposing an additional algorithmic complexity. Experts tend to solve problems by searching their knowledgebase for an appropriate solution. The simplicity of rule-based representation has made it the most commonly used form for knowledgebase systems according to [7]. The other advantage this form of representation has is its modularity as rules can be added or removed independently of other rules. Some of its disadvantages are related to performance as the number of rules increases and the difficulty in covering all the possible situations that exist in some domains. Many rule-based engines such as JESS[5] and SOAR[6] have dealt with the performance issue by using an optimised algorithm (i.e. RETE algorithm). The concern regarding the number of situations covered is minimized by the modularity of the approach and the ability to add situations when they arise if they have not been addressed earlier.

The ontology and rule-based representations seem to satisfy our two requirements and thus are the representations chosen for our approach. The next section describes how the two are combined to enable domain and reasoning knowledge to be stored.

3. Knowledge Creation

The process of creating the knowledge is done in two steps. First, the domain-ontology knowledge is entered followed by the rules which make use of the ontology knowledge. For instance, if the domain-expert wanted to enter a rule which relates to traffic investigation that states "If there is an injured person then the investigator has to attend to the injured person and request an ambulance from the operator", then the ontology knowledge has to have at least four entities: person, investigator, ambulance, and operator. Some of the entities need attributes to describe their status such as the person entity which needs an attribute describing his injury and possibly the ambulance entity needs an attribute to state that it is not out on another call. Some rules need to apply to a group of entities rather than individual ones. An example of such a rule is "if there is a person that is not at a safe distance from a danger source then move him away".

Furthermore, the data type of each attribute must be specified during the creation of the entity and an attribute might have further constraints such as minimum and maximum values or that it can only hold one of a number of predefined values such as injury type which can be one of the following values: none, slight, medium, severe, or deadly.

An ontology representation addresses the above representational concerns very well. The ontology creation lifecycle starts by creating the entity and placing it in its appropriate hierarchical location as

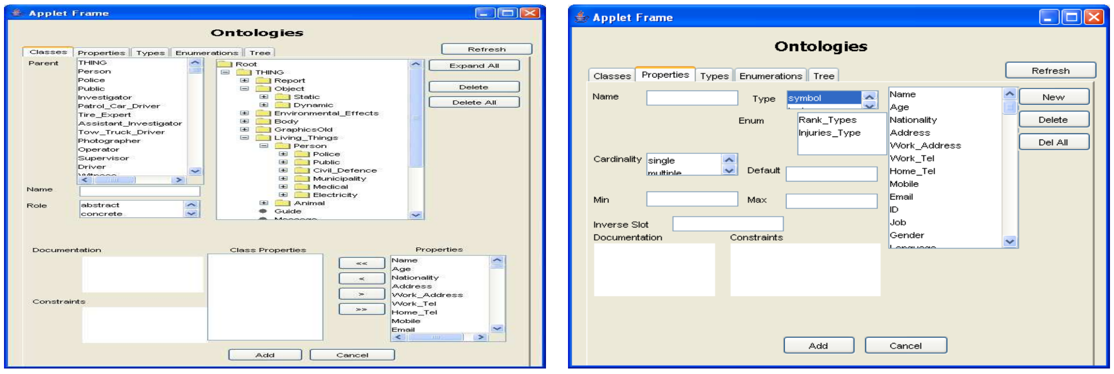


Fig. 2 Left: entities creation tab; Right: attributes creation tab.

[5] http://herzberg.ca.sandia.gov/jess/
[6] http://sitemaker.umich.edu/soar

shown in the left image in Fig. 2. This hierarchical capability addresses the group of entities concern raised by the second example rule which requires the system to provide the investigator with a hint listing all the persons that are not at a safe distance from the danger source. The system identifies all persons by searching for all the entities that descend from the ‘person’ entity as they are considered in the same group. The next step is to create the different attributes along with their constraint illustrated by the right image in Fig. 2. We followed Protégé 2.1’s list of constraints, which comprises of name, type, cardinality, default value, minimum value, maximum value, and documentation. The injury type issue mentioned earlier is dealt with by allowing an attribute data type to be an enumeration which holds the values permitted.

Once the entities and their attributes have been inserted the last step is to start adding the rules. Considering the two parts of a rule, condition and action, it seems logical to allow the domain-expert to create conditions and actions separately and then combine the two to make up a rule. This also maximises the reusability of the two parts as the same condition or action can be reused as part of other rules. Figure 3 shows the interface for inserting rules; it consists of three tabs: conditions, actions, and rules.

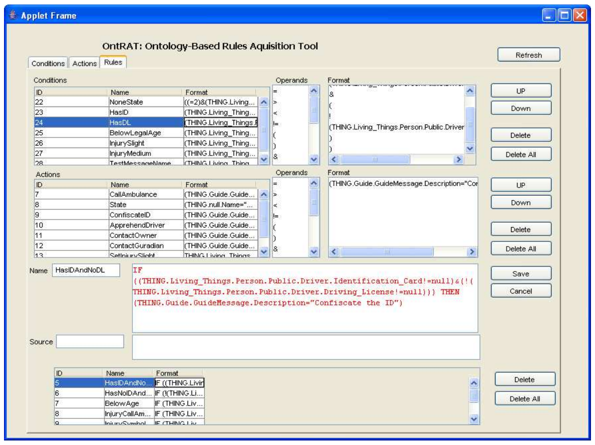
Continuing with the example rule of the injured person we show how the rule is inserted and formatted. The person, investigator, ambulance and operator entities are created by the domain-expert along with their attributes and constraints. The next step is to build the condition for the rule. The condition syntax consists of a variable followed by an operator and then another variable (<variable><operand><variable>). The variable on the left of the operator could be an entity attribute whereas the variable on the right could be a value or an entity attribute. This simple condition can further be combined with other conditions to form a more complex one (<condition><operand><condition>).

Therefore, the condition that states a person is injured can be formatted as “Person.Injury != none” where the dot is used to separate the entity from the attribute or another entity in a hierarchical relationship similar to an object-oriented declaration. This is then followed by the relationship operand between the two variables. The operand can be on one of: =, >, <, <=, >=, !=, &, and ||. The operands list has its own table in the database and can be extended further to include other operands. The value ‘none’ is taken from the injury enumeration.

The action syntax follows similar patterns to the condition. To avoid the need for operand precedence, parenthesis can be used, e.g. two actions are formatted as “((Investigator.hint= ‘Attend injured’) & (Investigator.hint= ‘call ambulance’))”.

4. Knowledge Usage

In this section we describe how the three primary stakeholders who need access to the knowledge – domain-experts, developers, and end-users – interact with the system to manipulate and use the knowledge to make a virtual environment (Fig.4) appear intelligent to end-users using one of the simulation engines in Fig. 1. The aim is to allow the end-user or trainee to navigate and interact with objects in the environment to aid him in solving a virtual traffic accident [8]. In this particular experiment we had a traffic accident scene involving two vehicles with one injured passenger and both drivers at the accident scene.



Conditions

ID	Name	Format
22	NoneState	((=2)&(THING.Living...
23	HasID	(THING.Living_Thing...
24	HasDL	(THING.Living_Things.F
25	BelowLegalAge	(THING.Living_Thing...
26	InjurySlight	(THING.Living_Thing...
27	InjuryMedium	(THING.Living_Thing...
28	TestMessageName	(THING.Living_Thing

Operands: = > < != ( ) &

Format:
```
&
(
!
(THING.Living_Things.Person.Public.Driver
)
)
```

Actions

ID	Name	Format
7	CallAmbulance	(THING.Guide.Guide...
8	State	(THING.null.Name="...
9	ConfiscateID	(THING.Guide.Guide...
10	ApprehendDriver	(THING.Guide.Guide...
11	ContactOwner	(THING.Guide.Guide...
12	ContactGuradian	(THING.Guide.Guide...
13	SetInjurySlight	THING.Living_Things

Operands: = > < != ( ) &

Format: (THING.Guide.GuideMessage.Description="Cor

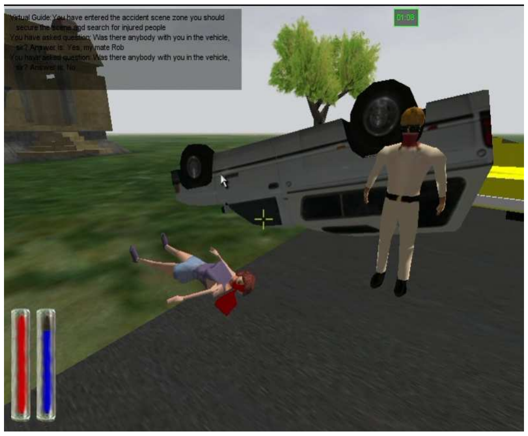
Name: HasIDAndNoDL

```
IF
((THING.Living_Things.Person.Public.Driver.Identification_Card!=null)&(!(
THING.Living_Things.Person.Public.Driver.Driving_License!=null))) THEN
(THING.Guide.GuideMessage.Description="Confiscate the ID")
```

Source:

ID	Name	Format
5	HasIDAndNo...	IF ((THING.Livin
6	HasNoIDAnd...	IF (!(THING.Li...
7	BelowAge	IF (THING.Liv...
8	InjuryCallAm...	IF (THING.Liv...
9	InjurySymbol	IF (THING.Liv

Fig. 3 Rules interface.



Virtual Guide: You have entered the accident scene zone you should secure the scene and search for injured people
You have asked question: Was there anybody with you in the vehicle, sir? Answer is: Yes, my mate Rob
You have asked question: Was there anybody with you in the vehicle, sir? Answer is: No

Fig. 4 Virtual traffic accident.

We wanted the trainee to be able to interact with the drivers to query their attributes and ask them questions. We also wanted the trainee to receive hints from the environment if he enters a dangerous area. The attributes query and questions requirement was addressed solely by knowledge represented by an ontology, whereas for the hints requirement we needed both the ontology and rules.

A meaningful vocabulary, familiar to the end-user, is used by the by the domain-expert for the attributes. The developer therefore only had to build a program to read and display the attributes without any extra manipulation. For the end-user this meant by clicking on any object in the virtual environment the list of its attributes are displayed and he can further click on any attribute to view its values thus enabling him to view the attributes as entered by the domain-expert. For the questions requirement, by design choice, the developer displayed the questions in a different panel on the end-user screen and hence the developer had to read the questions attribute separately and display it on a specific panel.

To enable the environment to provide the trainee with hints we needed a rule to monitor the user movements in the environment and once it detects that he has entered a specific zone to send the appropriate message. This meant that a zone entity should exist in the ontology knowledge and the chosen representation for that was a cubical representation with eight points. The rule then checked the trainee position in the environment against the eight points and sent messages accordingly. As shown in Fig. 1, currently the developer is involved in translating the rule to the logic-engine. However in the future this could be automated similar to the way the ontology knowledge is automatically translated.

5. Conclusions

The objective was to build a system that was accessible to all parties involved in building a domain knowledge that has a structure, a high-level abstraction, and the ability to reason about the knowledge. We have chosen ontology and rule-based representations to fulfil these attributes and shown how the three stakeholders can have access to the same knowledge and how it has been deployed in a virtual environment for training purposes. Because of the simplicity of the training example used the results can only be indicative of the viability of this approach and more complex problems need to be solved before getting a better picture of its extensibility and suitability. Additionally usability tests need to be conducted to evaluate the user interfaces. Future enhancements planned include multi-language support, and modules to export the ontology to a different format that already has some tools which can carry out verification tests on the knowledge.

References

[1] A. BinSubaih, S.C.Maddock, D.Romano, A Domain-Independent Multiplayer Architecture for Training, International Workshop in Computer Game Design and Technology, November 15-16th, Liverpool, UK, 2004, pp. 144-151.

[2] T. Morgan. Business Rules and Information Systems: Aligning IT with Business Goals. Addison Wesley, 2002, ISBN: 0-201-74391-4.

[3] B. Chandrasekaran , J. R. Josephson and V. R. Benjamins. What are ontologies and why do we need them?, IEEE Intelligent Systems, Jan/Feb 1999, **14**(1), pp. 20-26.

[4] N. F. Noy, R. W. Fergerson, & M. A. Musen. The knowledge model of Protege-2000: Combining interoperability and flexibility. 2th International Conference on Knowledge Engineering and Knowledge Management (EKAW'2000), Juan-les-Pins, France, 2000.

[5] H. Eriksson. Using JessTab to Integrate Protégé and Jess. IEEE Intelligent Systems **18**(2), 2003, pp. 43-50.

[6] P. Jackson. Introduction to Expert Systems. Addison Wesley Longman Limited, 1999, ISBN 0-201-87686-8.

[7] A. J. Gonzalez, D. D. Douglas, The Engineering of Knowledge-Based Systems, Theory And Practice. Prentice Hall International Editions, 1993, ISBN: 0-13-334293-X.

[8] A. BinSubaih, S. Maddock, D.Romano. Game Logic Portability. ACM SIGCHI International Conference on Advances in Computer Entertainment Technology ACE 2005, Games Technology and Design session, June 15-17th, Valencia, Spain. (to appear)