Purpose-Aware Reasoning about Interoperability of Heterogeneous Training Systems

Daniel Elenius, Reginald Ford, Grit Denker, David Martin, and Mark Johnson

SRI International, Menlo Park, California, USA
firstname.lastname@sri.com

Abstract. We describe a novel approach by which software can assess the ability of a confederation of heterogeneous systems to interoperate to achieve a given purpose. The approach uses ontologies and knowledge bases (KBs) to capture the salient characteristics of systems, on the one hand, and of tasks for which these systems will be employed, on the other. Rules are used to represent the conditions under which the *capabilities provided* by systems can fulfill the *capabilities needed* to support the roles and interactions that make up each task. An Analyzer component employs these KBs and rules to determine if a given confederation will be adequate, to generate suitable confederations from a collection of available systems, to pre-diagnose potential interoperability problems that might arise, and to suggest system configuration options that will help to make interoperability possible. We have demonstrated the feasibility of this approach using a prototype Analyzer and KBs.

1 INTRODUCTION

Much has been achieved in coaxing disparate resources to work together synergistically, but our ambitions often exceed the means at our disposal. Initiatives to lash together military training and testing systems are among the most ambitious. Since the 1980s, the means have evolved from dedicated engineered interfaces, to generalized interoperability middleware and protocols, and more recently to service-oriented architectures (SOA) and improvisational all-comer data access strategies such as the U.S. DoD Netcentric Data Strategy (NCDS) and the NATO Network Enabled Capability (NNEC). However, universal improvisational "plug and play" is still beyond our grasp. Successful interoperabilty is typically achieved only after lengthy planning, and it is not uncommon for apparent successes to exhibit subtle unacceptable anomalies.

For a nontrivial set of resources that have been developed independently to meet the unique needs of their sponsors/owners, it is impossible in general to answer the unbounded question "are these resources interoperable with one another"? However, when qualified as "are resources $R_1, \ldots, R_n$ interoperable with each other for purposes $P_1, \ldots, P_n$", the question becomes manageable. This paper describes the application of Semantic Web technologies to enable automated

"purpose-aware" reasoning about interoperability, as part of the Open Netcentric Interoperability Standards for Training and Testing (ONISTT) program.[1]

ONISTT is developing (a) ontologies to express the capabilities needed to perform mission-related tasks, (b) ontologies to express the capabilities available from prospective resources for executing those tasks, and (c) an automated reasoner/analyzer that can determine if the collective capabilities of some subset of candidate resources can satisfy the needs of a specific target mission. If multiple subsets can satisfy the needs, the analyzer ranks the relative goodness of each subset (with respect to relatively simple metrics). Because "goodness" of fit among resources is not always a simple "yes" or "no", the analyzer may give a qualified answer, leaving it to human judgment to say whether the level of interoperability is "good enough" or "the best obtainable." The analyzer can also suggest system configuration options.

In ONISTT proof of concept demonstrations, prototype KBs were populated with declarative information about the interoperability needs of particular training events, and the specific capabilities provided by operational and training/testing resources. Several alternative compositions of resources were presented to the prototype analyzer software. The analyzer correctly evaluated potential compatibilities and conflicts among the resources.

2 BACKGROUND

The training community distinguishes between *live*, *virtual*, and *constructive* training systems. In a live training system, real personnel and vehicles are augmented with instrumentation such as GPS trackers and firing simulators, so that combat situations can be trained as realistically as possible without the need to fire real ammunition. Virtual systems – used to train crews of tanks, aircraft, etc. – also involve real personnel, and the controls they use are often very close to those on the real vehicle or article, but their view of their surroundings is through computer-generated images. Constructive systems are done completely on a computer, with a human controller who decides where troops go, and so on.

For larger training exercises, there is often a need to connect different training systems, and the systems can be of all three kinds (i.e. live, virtual, and constructve). We refer to this as LVC training, and the set of systems used is called a *confederation*.

The problem with LVC training is that different training systems are usually not built to be used together. Although the "technical interoperabilty" of exchanging data among systems is often achieved, there are many ways in which "substantive interoperability" can fail to occur. We have found it useful to distinguish between four levels of interoperability, which we derived from the Levels of Conceptual Interoperability Model (LCIM) [1]:

[1] ONISTT is supported by the office of the Deputy Under Secretary of Defense/Readiness/Readiness and Training Policy and Programs (DUSD/R/RTPP). The views expressed in this paper are those of the authors and not necessarily those of DUSD/R/RTPP.

– Network interoperability. Common networking stack and medium. Systems can physically exchange digital messages.
– Syntactic interoperability. Common syntactic structure of messages.
– Semantic interoperability. Systems have a common understanding of the meaning of concepts used in communication.
– Behavior interoperability. Systems are compatible with regard to the actions they take on receiving messages, and the circumstances under which they send messages.

Many interoperability solutions have been implemented or proposed for training and similar systems [2–4]. However, none of these solutions encompass the whole range of interoperability problems. In particular, they usually only specify standards for networking and syntax, but fall short of a comprehensive semantics, and fail to address behavioral problems [1].

The end result is that one often does not know whether two systems will be truly interoperable until it has been tried. Much of the knowledge about interoperability and its problems resides in the minds of the engineers responsible for the systems. Currently, before any major LVC training exercise, a so-called BOGSAT (Bunch of Guys Sitting Around a Table) is summoned to work out interoperability issues. [5] This process is error-prone, costly, and time-consuming. Furthermore, the outcome of the process may not be stored for reuse in the future, leading to redundant efforts.

3 THE ONISTT APPROACH

In order to enable improvised training events on short notice, we need to automate a significant portion of the planning and setup of these events.

The solution we propose is based on Semantic Web technologies. Our approach is summarized in Figure 1. First (1), we develop what we call *referents* for all the training environments, tasks, infrastructures, and systems that are relevant to the problem. By "referent," we mean the most accurate and complete information available about the entity in question. Referents can include semi-formal models such as UML diagrams. These referents are then formalized into OWL ontologies (2). We discuss the ONISTT ontologies in more detail in Section 4. On the basis of these ontologies, the human planner defines an event and proposes a partial or full confederation for the event (3). We have developed a plugin to Protégé [6] to facilitate the formalization of the event. Then a piece of software called simply "the Analyzer" uses the data from the planner and the ontologies to verify the given confederation or generate a verified confederation, based on domain-specific rules and general reasoning technology (4). *Verified* means simply that the confederation passes all the interoperability tests that apply for the given purposes and circumstances. The Analyzer is described in more detail in Section 5. The Analyzer either returns a verified confederation (5a) and *configuration artifacts* (to be explained in following sections), or notifies the planner of what went wrong (5b), the more common situation. The Analyzer can report anything from minor warnings to major roadblocks to a successful event. At this point the planner can take corrective action and submit a new

proposal to the Analyzer, starting the process over. Alternatively, the planner may choose to live with the problems that the Analyzer found.

Our goal is that the output of this process should be at least as good as the output of a traditional BOGSAT, while being cheaper, less time-consuming, and more reusable.

4 ONTOLOGIES

4.1 Ontology Language and Structure

We use OWL to express our ontologies. The primary reasons for selecting OWL are pragmatic rather than technical (e.g., we do not make use of an OWL DL subsumption-based reasoner, see Section 5). For our approach to be successful, other groups will need to adopt it, and mature ontology engineering tools will be required. We have created some prototype ontologies, but in the long run, we cannot ourselves encode all the knowledge pertaining to this domain. Therefore, we need a language and tools that other groups can readily pick up and use. The DoD and others are converging on OWL for ontology expression. Although fully mature OWL engineering environments are not yet available, open-source and commercial tools are growing in number and capability.

Our ontology set is highly modularized, relying heavily on the OWL import mechanism. The top-level ontology, onistt.owl, has a number of relatively "naked" concepts and some properties between them. Each concept is elaborated in one or more special ontologies. To describe the full richness of the training domain, we also need ontologies of training systems, communication standards, virtual terrain, military vehicles and weapons, and so on. We have made a start at the ontologies we found necessary for the scenarios we have worked with. In total, we have about 60 relatively small prototype ontologies. The intent is that different organizations should be responsible for fleshing out their own ontologies.

4.2 Ontology Overview

The ONISTT ontology, as shown in Figure 2 has three complementary parts that compose a *Deployment*:

- An *Exercise* has *Task* objectives (i.e., "purposes"), from which an assemblage of needed capabilities is derived.
- A *Confederation* is a collection of *Resources* whose individual capabilities may be composed to satisfy the sum of the capabilities needed.
- A set of *Assignments* match the capabilities provided by individual confederation resources with specific capabilities needed to conduct the exercise.

Sections 4.3 to 4.5 explain how Exercise, Confederation, and Assignment properties and concepts are tailored so that the Analyzer, as described in Section 5, can satisfy the objectives identified in Section 3. Although the ontology design reflects the intended use of our tool suite in exercise planning, we think that similar purpose-resource matching ontologies could be developed for other domains. Also, many of the imports are prototype generic ontologies we de-

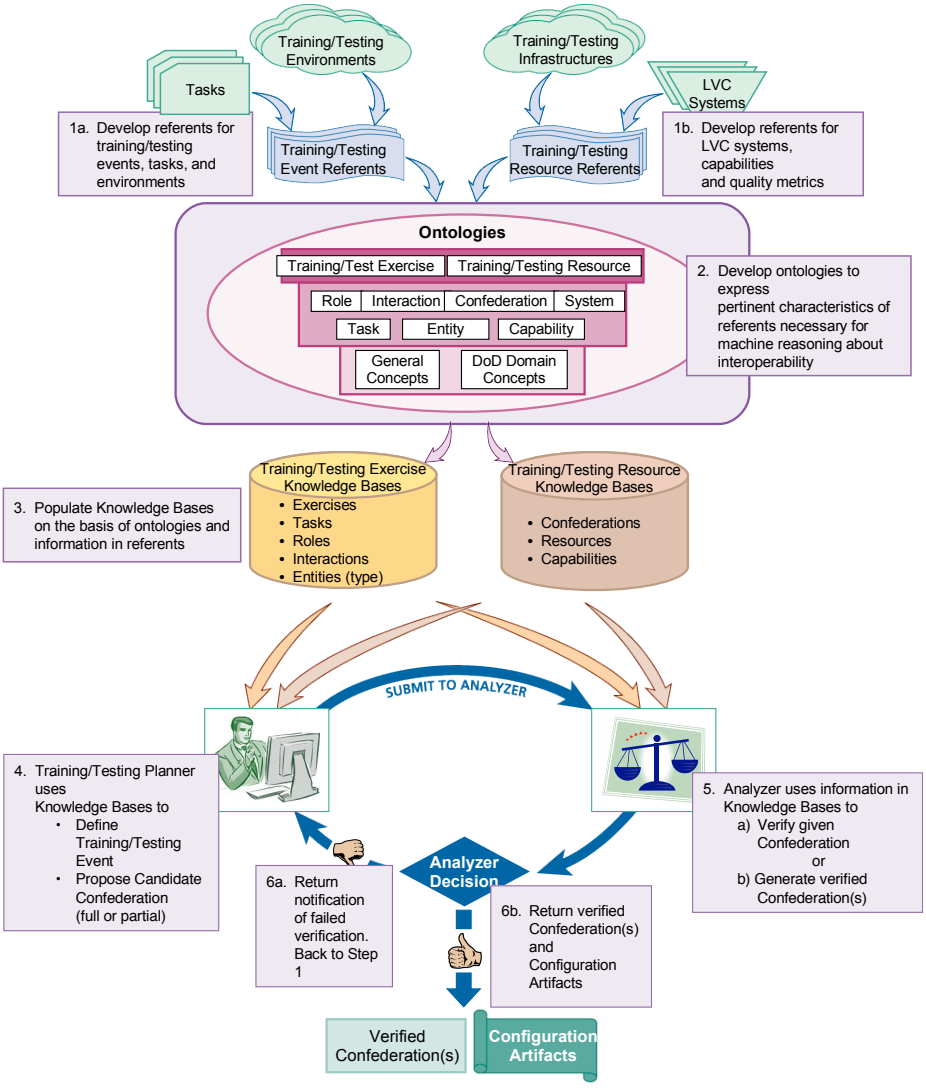

Fig. 1. The ONISTT approach and methodology.

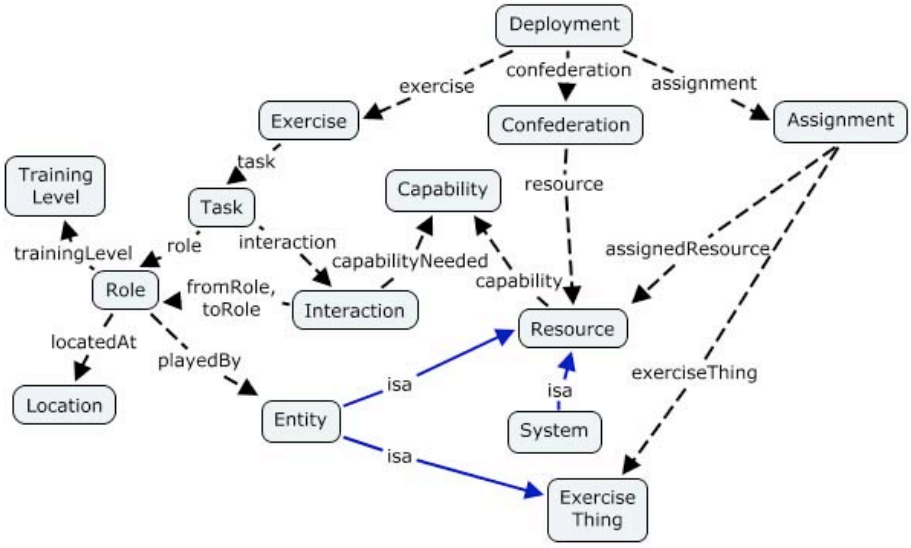

Fig. 2. Top-level ONISTT ontology concepts

veloped for technical domains, for example the spatial ontologies discussed in Section 4.4.

4.3 Purpose Ontologies

The ONISTT "purpose" ontologies are shown on the left in Figure 2. The purpose of a training *Exercise* is to improve the proficiency of military forces in the conduct of one or more *Tasks*. Joint training tasks (e.g., Joint Close Air Support) are defined in the Universal Joint Task List (UJTL). Responsibility for executing a task is distributed among several *Roles* (e.g., forward observer). A Role is *playedBy* an *Entity*. The UJTL describes friendly force roles. However, a training exercise must provide its own opposing force, and so surrogate hostile force roles are included in ONISTT KBs.

Our ontology includes a prototype taxonomy of Role subclasses and their associated properties and restrictions. For example, the playedBy property of the *AirborneOrdnanceDeliveryRole* has `allValuesFrom` and `someValuesFrom` restrictions that require individuals to belong to the *AirPlatform* class.

Each role in a task is responsible for performing certain actions. A complete description of tasks would include actions that can be assigned to a single role. However, the scope of ONISTT is interoperability, and therefore the only actions defined in the ontology are *interactions* among roles. The initiator of an interaction is designated by the *fromRole* property, and the other end is the *toRole*. Each interaction is associated with one or more needed *Capabilities*. Interactions between opposing forces are often the most critical to planning a training exercise because they require the support of special training capabilities, for ex-

ample communications and simulations needed to effect a weapons engagement. However, training environments often differ from real operational environments in ways that require some interactions even among friendly force roles to be supported by auxiliary training system resources.

In defining an exercise, the planner may designate different *TrainingLevel* objectives for different roles. The particular interactions that are required between roles, and the required qualities and characteristics of the interactions, may depend on the training level. An exercise scenario defines other constraints on roles, such as *Location*.

4.4 Resource Ontologies

The ONISTT "resource" ontologies, which specialize the *Resource* class shown in the middle of Figure 2, are designed to support KBs that describe (1) the capabilities of individual resources, and (2) the bundling of resources.

Since most systems and other assets used in training exercises are multipurpose and adaptive, ONISTT ontologies are designed to describe resources fully, apart from any intended application. The Analyzer software selects facts from the "resource" KBs to determine whether a particular combination of resources is capable of performing a particular set of tasks in a particular context, as described in Section 4.5.

Most resources in LVC training are not accessible atomically but are bundled with other resources. The highest-level collection of resources in our ontologies is the *Confederation*. A Confederation is typically a loose and temporary aggregation of *Resources* – systems, infrastructure, and other assets – that have a more stable, though not necessarily fixed, identity. Resources can have multiple *subresources*, which in turn have subresources, to an arbitrary number of levels. Dependencies among subresources constrain whether they can be used in a mix-and-match fashion or are coupled together.

The main subclasses of Resource are *Entity* and *System*. An entity is an exercise participant, for example an F/A-18 aircraft. Training systems are often very complex, with multiple subsystems, which are recorded in KBs as subresources whose range is also the System class. An example is live training instrumentation that tracks the position of exercise entities, performs weapons effect simulation when entities engage each other, records data for After Action Review (AAR), and so on.

Each Resource has one or more *Capabilities*. One subclass of interest to the ONISTT domain is *Representation Capability*, for example the ability of constructive modeling and simulation (M&S) to simulate an F/A-18 entity. Another subclass is *Communication Capability*, for example the ability of a training system to communicate with other systems using the Test and Training Enabling Architecture (TENA). Another is *Terrain Capability*, for example the ability to determine line of sight between two earth locations.

Quality and type characteristics are specific to Resource types. For example a Representation Capability has a *motionModel* property. A *Constructive Representation Capability* is restricted to motion models of the *Simulated Mo-*

tion Model. One of the quality characteristics of a simulated motion model is a boolean property indicating the physical realism of a simulated entity's motion in turns (smooth or jerky). A few relatively simple quality metrics were used for the initial feasibility demonstration. One of the major challenges for ONISTT is expressing qualities and characteristics of capabilities in a standard way so that the capabilities provided by resources can be compared with the capabilities needed [7–9].

An example of standard definition of capability characteristics and qualities is shown in Figure 3. Many training systems exchange Time-Space Position Information (TSPI). Because the shape of the earth is irregular, it is difficult to express geographic coordinates precisely and unambiguously, and many mathematical schemes have been developed. A recepient of TSPI must know the spatial reference frame of the data to interpret it correctly. The ISO 18026 Spatial Reference Model standardizes geographic reference frames. We translated the ISO 18026 spatial reference frames most commonly used in training systems to an ontology. If an interaction requires two systems to exchange TSPI data, the Analyzer can examine if they use the same reference frame or, if not, whether they have a capability to translate between the two frames. If translation is required, the Analyzer can determine if the translation will be perfect, or if certain attributes, such as line-of-sight calculation, may be distorted.

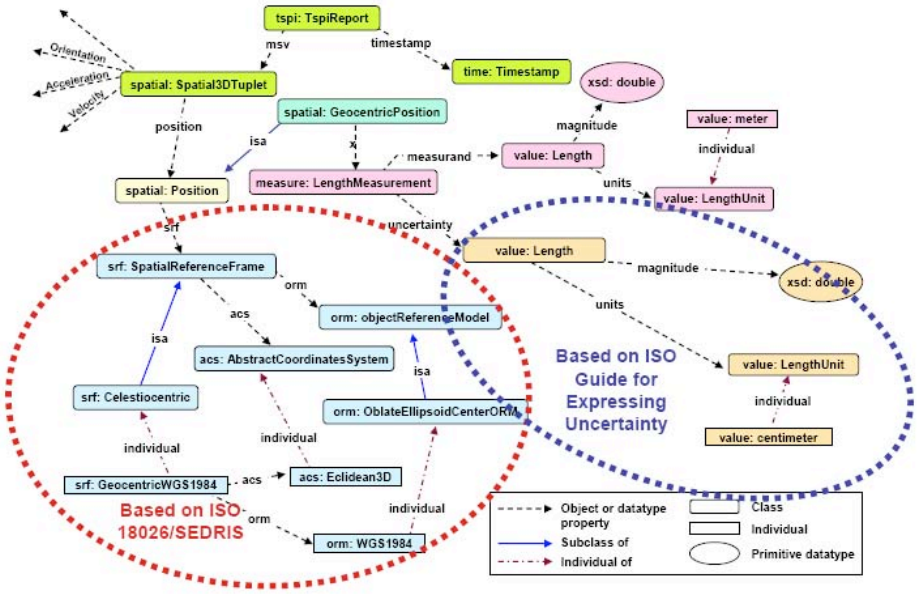

Fig. 3. Slice of the TSPI report ontology.

We developed approximately 20 prototype KBs describing individual systems and capabilities to support the feasiblity demonstration. A full ONISTT operational capability will require hundreds. Fortunately, a comprehensive description of each resource and capability is not necessary. The level of detail needed in KBs is limited to facts that are directly relevant to assessing interoperability among resources.

4.5 Assignment Ontologies

The "Assignment" ontology, as shown on the right in Figure 2, connects the "purpose" and "resource" ontologies. The objective of exercise planning is to specify a valid *Deployment* that defines a *Confederation* of resources to meet the needs of a particular *Exercise*. An individual Exercise is a set of individual Tasks, Roles, Interactions, and Entities selected from KBs developed in accordance with the ontologies explained in Section 4.3. An individual *Confederation* is a pool of systems and other resources that are selected from KBs developed in accordance with Section 4.4. Individual *Assignments* assign one or more Confederation Resources to each Entity defined in the Exercise. The Analyzer software determines whether the assigned resources have all the requisite capabilities and capability qualities and characteristics needed to support all Interactions among all Entities.

5 ANALYZER

We mentioned in Section 3 the use of Analyzer software to provide feedback on the suitability of a given confederation. Here, we describe the implementation of this software, which brought up some issues of more general concern in using Semantic Web technologies in the real world.

While the ontologies provide the background knowledge of the problem domain, the Analyzer provides the computational side of the automation. Its job is to look at the information provided and draw conclusions according to a set of *rules*.

We wanted to frame the operation of the analyzer as a problem of logical deduction, as this would give us a clear semantics of what the analyzer does. We considered (and tried) several different possibilities.

The most natural approach would be formulate the operation of the Analyzer as an OWL subsumption check. This would allow us to use any OWL DL reasoner right out of the box. However, we found it impossible to formulate the problem in this way, mainly because OWL DL is very restrictive with the use of quantifiers and variables.

Another approach was to use OWL augmented with SWRL [10] rules. However, SWRL also proved insufficient to express the operation of the Analyzer. This is explained in more detail below

A third approach was to translate the OWL KB to First-Order Logic (FOL) and axiomatize the operation of the Analyzer in FOL. We tried this (with the SNARK[2] theorem prover), but we found this to be too slow and sensitive to

[2] `http://www.ai.sri.com/snark/`

small changes in the problem formulation (as is often the case with applications of FOL theorem proving).

In the end, we decided to write the Analyzer as procedural code. We needed a tight integration of procedural code (the Analyzer) and declarative content (the KB). Prolog is a natural choice in this kind of situation. A large fragment of OWL, called Description Logic Programs (DLP) [11], can be readily translated to Logic Programs [12] (the logical underpinnings of Prolog). Prolog can also be used as a programming language for writing procedural code. In particular, we chose to use XSB Prolog,[3] in order to avoid the well-known problems that ordinary Prolog has with recursive structures such as equivalent classes or properties.

The Analyzer implementation consists of a pair of software components: A *Translator* that translates from the OWL+SWRL KBs into XSB Prolog, and the *Analyzer Core* that runs domain-specific tests on the information in the knowledge base.

5.1 Translator

Fortunately, there is already a "standard" way [11] to translate a fragment of OWL DL, called DLP, to Logic Programs. It is also straightforward to extend this translation to handle SWRL rules, because these rules are just Horn clauses, directly expressible in Logic Programs (and thus in Prolog). We will describe some of the salient features of our translation approach in the following.

A naive translation to Prolog (as done by for example the `dlpconvert` tool, which is part of the KAON2 toolset[4]) encodes OWL classes as unary Prolog predicates, OWL properties as binary predicates, and OWL axioms to Prolog rules.

However, we have chosen a different encoding, with several advantages that will be explained shortly. Our target Prolog statements include only two predicates, *inst*/2 and *value*/3, with the intuitive meaning that inst(x,y) means that x is an instance of y, and value(x,y,z) means that x has value z on property y.[5] In this encoding, all class, property, and instance names appear as Prolog *terms*. We may say that these entities are *reified*, as they appear as objects in the target language.

This encoding allows more types of queries to be answered. For example, we may ask for all classes that John is an instance of, `?- inst(John,X)`, with answers like `X = person; X = animal; X = thing`. Note that the answers are not just asserted instance relationships, but also all relationships that can be inferred according to the semantics of the DLP+SWRL fragment. Another type of query is to ask for all known or inferred property values of an instance, `?-`

[3] `http://xsb.sourceforge.net/`

[4] `http://owltools.ontoware.org/`

[5] We could have encoded everything using the value predicate by making "type" a property. However, we opted for a more intuitive encoding that avoids special properties like `rdf:type`.

`value(john,Prop,Val)`, and get answers like `Prop = sibling Val = Dave; Prop = sibling Val = Alice; Prop = age Val = 23`.

This approach gives us more expressive power in formulating queries. Not just instances, but also classes and properties, become objects that we can refer to in queries. For example, we can use a class as the first or third argument to the value predicate. This is usually called classes-as-instances, or just reification, and is a feature of OWL Full. The jury is out on whether this is actually a desirable feature in a language. One view is that a perceived need for this feature indicates that something is wrong with the ontology. This may be true in a "pure" OWL DL ontology that is designed with OWL DL reasoning in mind. However, for the kinds of queries we do, we have found that classes-as-instances is quite a useful feature in a few select places in our ontologies.

5.2 Analyzer Core

Once the OWL+SWRL knowledge base has been translated, it can be loaded into our Prolog engine, and used by the Analyzer Core.

The Analyzer Core is a piece of code, currently a few hundred lines, that is also written in XSB Prolog. While the translated knowledge base uses XSB in a declarative way by just stating the facts, the Analyzer Core is a procedural program that queries the knowledge base in various places. This is a very flexible approach to integration of programming and reasoning.

More specifically, the operation of this component is as follows. It looks at each pair of Roles that have an Interaction between them. It then compares the Capabilities needed for that type of Interaction with the Capabilities provided by the Resources that are used to represent the Role. The resources may be sufficient, insufficient, or somewhere in between. In the real world, the latter is usually the case. Therefore, the Analyzer generates warnings for conditions that may be problematic but not fatal. Each condition is encoded as a rule in the Analyzer.

Let us look at one of these conditions, *unrealistic_motion*. This condition occurs when we have an interaction requiring line of sight (LOS), for example, a DirectFire interaction, *and* the fromRole is represented by a virtual system (such as a tank simulator). Under these conditions, we require that the toRole has smooth motion modeling. An example of where this is not true is when the toRole is represented by a constructive system where units move betweeen waypoints without smoothing out the curves (i.e., they have infinite angular acceleration). This causes the users of the virtual system to see units "jumping around" in an unrealistic way on their 3D displays. This leads to a loss of realism, and is also a "fair fight" issue, since it is hard to target units that are moving in a way that is physically impossible. The Prolog encoding of this rule is as follows:

```
% I - Interaction
% FRes - set of resources that represent fromRole
% TRes - set of resources that represent toRole

realisticMotion(I,FRes,TRes) :-
```

```
realisticMotion_cond(I,FRes), !, realisticMotion_constraint(TRes).

realisticMotion(_I,_FRes,_TRes).

realisticMotion_cond(I,FRes) :-
inst(I,'interaction:LOS_Interaction'),
member(FS,FRes),
value(FS,'onistt:subresource',FSs),
value(FSs,'system:representationCapability',FRC),
inst(FRC,'system:VirtualRepresentationCapability').

realisticMotion_constraint(TRes) :-
member(TS,TRes),
value(TS,'onistt:subresource',TSs),
value(TSs,'system:representationCapability',TRC),
value(TRC,'system:motionModel',MMT),
value(MMT,'motion:smoothMotion',true).
```

If the realisticMotion procedure fails, then the condition occurs (i.e., if we do not have realisticMotion, then we have unrealistic motion). The top-level rule is an implication that must be satisfied; *if* realisticMotion_cond holds, *then* realisticMotion_constraint must also hold.

Several things are worth pointing out in this example.

- The rule is stated in a *positive* way – unless we can prove the absence of a warning condition, we assume that the condition holds. This means that a warning condition cannot be avoided simply because of lack of information.
- We see that the Analyzer queries the knowledge base in many places, that is, it invokes the *value* and *inst* predicates. Each of these calls can involve arbitrary DLP reasoning. For example, the onistt:subresource property is transitive, so the value calls using this property will return a transitive closure. This is what we meant by a flexible integration of programming and reasoning, above.
- Different rules have different sets of arguments. In this case, we need the Interaction, the from-resources, and the to-resources. Other rules are slightly different. Also, some rules have return values. For example, the rules to check whether two roles can communicate also return the communication path (we call this a *configuration artifact*).

Other examples of conditions are lack of training system or tactical communications (where needed) or uncorrelated terrain data, which can cause problems like tanks hovering in the air, or airplanes flying through mountains.

The set of rules to check for problematic conditions is meant to be extensible. Ideally, one should be able to specify these rules in a declarative way – for example in SWRL – so that they can easily be inspected, edited, and so on. This was also our original approach. However, we found that the expressiveness of SWRL was not sufficient to encode the rules. To see why this is so, first consider the requirements for the Analyzer. The Analyzer is basically a *function* that takes one argument, the Deployment to check, and returns the results of the analysis, that is, warnings for some of the interactions in the Deployment, and configuration artifacts. These results could be encoded as ontological objects

(e.g., a Warning class, with subclasses for different types of warnings), and we would then like the Analyzer to create and return the appropriate instances. However, SWRL rules cannot create new instances, or indeed return any type of structured data, unless the data is part of the "input" arguments to the rule. We *can* do this in Prolog, by using compound terms, but SWRL is Datalog (i.e. function-free), and thus it does not have compound terms. One option that we plan to explore for the purpose of making the Analyzer rules more transparent is Functional RuleML.[6]

6 EVALUATION

Our work on the ontologies as well as the analyzer has been driven by a number of use cases [13]. Each use case consisted of an exercise and several different confederations (and assignments) for that exercise. Working up the use cases forced us to develop the ontologies of the systems and tasks involved to a considerable level of detail. The cases were inspired by real-world exercises and systems with which we had previous experience, and were designed to provide a realistic level of complexity. Some of our use cases were such that we knew what the results should be, and we could verify that the analyzer came up with those results. We also developed new and more complex scenarios, where we did not know the answers beforehand. The analyzer returned useful warnings and configuration artifacts, such as mediated communication paths. We have not yet tried the analyzer on a real-world exercise ahead of time. Finding an appropriate exercise to work with is high on our agenda.

There is much to gain by applying Semantic Web technology to our domain, in terms of automation, reduction of costly labor, and new functionality. At the same time, the approach requires a large one-time investment in ontology development before it can be widely adopted. It is our contention that the benefits are large enough to motivate the costs, especially since the ontologies can be reused in many different contexts.

7 CONCLUSIONS

We have described a novel approach by which software can assess the ability of a confederation of heterogeneous systems to interoperate to achieve a given purpose. This approach uses ontologies and KBs to capture the salient characteristics of systems, on the one hand, and of tasks for which these systems will be employed, on the other. Rules are used to represent the conditions under which the *capabilities provided* by systems can fulfill the *capabilities needed* to support the roles and interactions that make up each task. An Analyzer component employs these KBs and rules to determine if a given confederation will be adequate, to generate suitable confederations from a collection of available systems, to pre-diagnose potential interoperability problems that might arise, and to suggest system configuration options that will help make interoperability possible.

[6] `http://www.ruleml.org/fun/`

Representing the capabilities of systems and reasoning about interoperability are notoriously difficult problems, in their general forms. Solutions to these problems have potential value in a number of domains and applications. In military settings, such as complex training exercises, it is often a top priority to minimize the engineering effort, and maximize the flexibility, associated with the deployment of "improvised" systems of systems. The work described here demonstrates a promising way forward. A key enabler of this approach is the explicit representation of purpose (i.e., tasks, roles, interactions, and the capabilities required for their fulfillment). We have found that assessing interoperability for a given purpose is considerably more manageable than general, unconstrained forms of the interoperability problem.

REFERENCES

1. Tolk, A., Turnitsa, C.D., Diallo, S.Y.: Implied ontological representation within the levels of conceptual interoperability model. International Journal for Intelligent Decision Technologies (IDT) **2** (2008) *To appear.*
2. Distributed Interactive Simulation Committee of the IEEE Computer Society: IEEE standard for distributed interactive simulation – application protocols (1998) IEEE Std 1278.1a-1998.
3. Simulation Interoperability Standards Committee of the IEEE Computer Society: IEEE standard for modeling and simulation (M&S) high level architecture (HLA)-framework and rules (2000) IEEE Std 1516-2000.
4. Hudgins, G.: DoD ranges interoperability reuse achievable through the Test and Training Enabling Architecture, TENA. In: International Telemetering Conference. (2006) 23–27
5. Johnson, M., Ford, R., Shockley, J., Giuli, R., Oberg, S., Beebe, M.: Integration of CCTT and JCATS in an LVC exercise. In: Simulation Interoperability Workshop. (2004) 04E-SIW-066.
6. Knublauch, H., Fergerson, R., Noy, N., Musen, M.: The Protégé OWL plugin: An open developoment environment for semantic web applications. In McIlraith, S., Plexousakis, D., van Harmelen, F., eds.: Proc. 3rd Intern. Semantic Web Conference (ISWC 2004), Hiroshima, Japan, November 2004, Springer (2004) 229–243 LNCS 3298.
7. RPG Special Topic: Fidelity. Technical report, Defense Modeling Simulation Office (DMSO) (2000)
8. Davis, P.K., Anderson, R.H.: Improving the composability of DoD models and simulations. Technical report, RAND National Defense Research Institute (2003)
9. Kasputis, S., Oswalt, I., McKay, R., Barber, S.: Semantic descriptors of models and simulations. In: Simulation Interoperability Workshop. (2004) 04F-SIW-070.
10. Horrocks, I., Patel-Schneider, P.F., Boley, H., Tabet, S., Grosof, B., Dean, M.: SWRL: A Semantic Web Rule Language Combining OWL and RuleML (2004)
11. Grosof, B.N., Horrocks, I., Volz, R., Decker, S.: Description logic programs: Combining logic programs with description logic. In: Proceedings of the 2nd International Semantic Web Conference (ISWC2003). (2003)
12. Loyd, J.W.: Foundations of Logic Programming (2nd extended ed.). Springer Verlag, New York (1987)
13. Ford, R., Hanz, D., Elenius, D., Johnson, M.: Purpose-Aware Interoperability: The ONISTT Ontologies and Analyzer. In: Simulation Interoperability Workshop, 07F-SIW-088, Simulation Interoperability Standards Organization (2007)