An Ontology for Resource Sharing

Kurt Rohloff, Joseph Loyall
BBN Technologies
Cambridge, MA
{krohloff,jloyall}@bbn.com

Abstract — **We describe an ontology for resource sharing in integrated systems. We call this ontology the "Resource Sharing Ontology." This ontology addresses one of the main challenges for system and service integration: the management of resource sharing interactions. These interactions, whether explicit or implicit, are difficult to model and manage, but they are critical for safe and efficient system designs. Our resource sharing ontology also covers performance assessments on resource sharing for online and offline control and diagnosis. We discuss ontology extensions for specific resource sharing scenarios such as the assessment of resource sharing complexity. We provide examples of using the ontology to model applications, such as an RLC circuit and for the assessment of a resource contention complexity metric for the maintenance of a prototype hybrid vehicle.**

Keywords- ontology; system integration; performance assessment

I. INTRODUCTION

One of the main challenges, if not the main challenge, for system and service integration is the management of sometimes complimentary, sometimes antagonistic subsystem interactions through resource sharing. Systems, whether cyber, physical, cyber-physical, or otherwise, are comprised of resources and system actors that interact with and through the resources. Shared resource interactions, whether explicit or implicit, are difficult to model and manage, but they are critical for safe and efficient system designs. These interactions must be considered in the design, development, and testing phases of system creation.

Although there has been a tremendous growth in the use of ontologies to facilitate systems and service integration in general [1][5][9][10][11][15], there has been little work on general ontologies for the key challenge of resource sharing as needed for offline or online resource allocation and reallocation [4][13][14]. When resource sharing interactions are not considered early in system creation, e.g., during the design phase, then they must be tested for during the system verification and validation phase, which is costlier, or there is a risk that they will be discovered after system deployment, where unanticipated resource interactions and contention that affect the system operation or performance can lead to even more costly system failure or retrofitting.

Sponsored by Defense Advanced Research Projects Agency, Program: META, Issued by DARPA/CMO under Contract No. HR0011-10-C-0108. The views expressed are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government. Distribution Statement "A" (Approved for Public Release, Distribution Unlimited).

In this paper we describe an OWL ontology [8] for resource sharing in integrated systems. We call this ontology the "Resource Sharing Ontology." We design our ontology to be sufficiently general to cover cyber, physical and cyber-physical systems. Resources may include, for example, engine power, fuel, communication bandwidth and cooling capacity in modern automobiles which are classic examples of cyber-physical systems.

The design of our resource sharing ontology is motivated by prior work designing automated resource allocators for distributed computing systems [12] and developing methods to reduce the design time for large cyber-physical systems.

We identify general classes of system actors that provide, consume or regulate resources. We identify general categories of attributes of shared resource languages that should be captured in the ontology. We design the ontology to be extensible and discuss ontology extensions for specific resource sharing scenarios such as the assessment of resource sharing complexity. This resource sharing ontology and its context-specific extensions are used to build composed models of systems components that interact both through directed communications and implicit and explicit resource sharing.

Although there has been work on ontologies for interoperable system design [10], there has been little specific focus on shared resource interactions and assessments for distributed systems. A formal ontology intended as the basis for building interoperable and reusable knowledge systems is provided in [1]. This work is generally more qualitative than quantitative and provides a vocabulary for modeling the structure and behavior of systems without discussion of performance assessment necessary for diagnosis, reconfiguration, tuning, etc. The work in [3] is more qualitative and provides an ontology for engineering mathematics without a focus on resource interactions. Similar work on ontologies for integration of manufacturing systems that incorporates resource sharing concepts is discussed in [5], [7], [9] and [15], but generally at a higher level than the ontology discussed herein. The ontology in [5] discusses resources with a focus on human, machine and tool resources for manufacturing without focusing on resource consumption. Machines and materials are considered as resources in [9].

The remainder of this paper is organized as follows. In the section immediately following we introduce the resource sharing ontology's design objectives. Section 3 introduces the ontology, its classes, object properties and data properties. Section 4 discusses the extensibility of the ontology for domain specific use with a resource contention metric. In Section 5 we

discuss two application examples using the ontology for resource sharing in RLC circuits and a hybrid HMMWV model. We close with conclusions in Section 6.

II. Ontology Objectives

Our ontology design is motivated by our understanding that actors operate through integrated systems and services to either consume, provide or regulate resources. These system actors include resource consumers, resource providers and resource allocators. Resource consumers use resources – for example an internal combustion engine consumes fuel, or network cards use bandwidth.

Resources may be transformed and disappear due to consumption (such as fuel), or resources may be used, but not disappear due to use (such as bandwidth). Resource providers generate resources that are consumed, e.g., an alternator attached to an engine generates electrical energy or a sensor may provide information about object detections. Resource allocators decide which resources are used by which consumers. For example, an engine control module determines the rate of fuel flow to an engine and a TCP implementation determines the use of bandwidth used by nodes communicating on an intranet.

System actors often need to make measurements of system conditions to assess the behavior and performance of resources and actors. These assessments are used by the system actors to alter their behavior, or by users to assess system health. For example, a resource allocator such as an engine control module uses measurements including engine speed, temperature, and atmospheric density to assess engine performance and allocate fuel to an engine. Similarly, TCP uses measurements of packet sequence numbers to assess congestion. In an example we discuss below in Section 5, we assess a metric called resource sharing complexity over various measurements to assess the propensity for resource contention.

The system actors, measurements, and assessments do not necessarily operate continuously like a centrifugal governor operating on a steam engine. Most often system actors, measurements, and assessments are implemented digitally and update either on a clock cycle or when driven by external events. As such, system interactions, measurements and assessments need to be modeled and coordinated with respect to event occurrences which may include clock ticks.

Depending on application, system actors (such as resource consumers, resource providers, and resource allocators) primarily interact through more than shared resources. Components could interact through directed communication, but these kinds of engineered/designed interactions are implemented through resource sharing interactions such as the use of communication buses. Our insights in this document are driven primarily by experience and published reports on developing and using resource sharing models to architect component interactions in information management systems [2], [4], [6]. Based on our experience [12], [13], [14], (explicit or implicit) shared resource interactions are difficult to capture and express. We see the need for a resource sharing interaction language that would be extensible and compatible with component modeling languages.

Selecting the attributes for a component model language for shared resources requires the selection of an appropriate level of abstraction. Although users of modeling languages may sometimes want or need to understand the inner operations of components, it is often sufficient to exclusively model aspects of component interactions, even when these interactions do not occur through explicitly specified interfaces. Furthermore, it is necessary to abstract from many of the details of the inner operations in order to reduce and manage the complexity involved in system modeling, design, and development. Since components at a given level of abstraction interact explicitly and implicitly through shared resources, a component model language for shared resources needs to capture the relevant properties of resource interactions. We focus on resource interactions because many component interactions can be described as interactions by the components through shared resources.

Based on the above motivation, we identify four general categories of attributes for shared resource model languages that should be considered in a resource sharing ontology:

1) Resource provisioning attributes: these attributes cover how the resources are allocated.

2) Resource availability attributes: these attributes cover how the availability of the resources may change after provisioning.

3) Resource consumption attributes: these attributes cover how the resources are consumed by component operation.

4) Resource assessment attributes: these attributes cover how the consumption of the resources are typically evaluated.

In the subsections immediately following we describe these categories. The attributes and categories we discuss are intended to be representative, and are by no means exhaustive or exclusive for all application domains. Our goal is to provide an ontological framework that is sufficient for large classes of systems. We intend for our taxonomy of attributes and the ontology to be further extended and customized for specific applications and systems.

All of the attribute categories we list include some aspect of resource constraints. In fact, most of the attributes describe some aspect of constraints on the behavior and use of the resources that need to be expressed in the resource models. These constraints limit the behaviors that need to be accounted for in composing model components through resource interactions.

A. Resource Provisioning

The resource provisioning attribute category contains attributes that describe how resources are allocated. These attributes include:

1. Resource Shared: This attribute captures whether the resource is shared or private.

2. Allocation Decider: This attribute captures who decides on the allocation of resources. Possible values include the system designer, system user, or the system for autoconfiguring systems.

3. Allocation Mechanism: This attribute captures how the allocation is decided, whether by directed allocation from a human-in-the-loop (such as a designer or user), a supervisory controller, or a collaborative resource allocation system in peer-to-peer environments.

4. Allocation Dynamism: This attribute captures whether the allocation is performed only once, or whether reallocation occurs during runtime.

5. Reallocation Trigger: When resource allocations are dynamic, this attribute captures how reallocation is triggered – whether from a clock tick, a feedback control mechanism, or driven by an external event like changes in missions.

B. Resource Availability

The resource availability attribute category contains attributes that describe how resource availability changes over time and how resource consumers and allocators observe those changes in availability. These attributes include:

1. Resource longevity: This attribute captures whether the resource has permanence (like physical space), it is consumed constantly (like time), or its consumption is a function of usage properties (like how fuel is consumed based on performance demands.)

2. Resource availability observability: This attribute captures whether the resource availability is directly observable, partially observable, or unobservable to various entities including consumers, providers and allocators.

C. Resource Consumption

The resource consumption attribute category contains attributes that describe how resource consumption occurs. These attributes include:

1. Resource consumption predictability for consumer: This attribute captures how predictable resource consumption is for consumers – whether it is schedulable (such as fuel availability due to consumption), partially schedulable (such as ammunition for weapons), or unschedulable (such as armor plating.)

2. Resource consumption predictability for allocator: This attribute captures how predictable resource consumption is for allocators.

3. Resource consumption controllability: This attribute captures whether the resource consumption is fully controlled (like fuel usage), partially controlled (like heat dissipation capability), or uncontrollable (like time.)

4. Resource usage correlation: This attribute identifies when consumption of particular subsets of resources are positively or negatively correlated.

D. Assessment of Resource Usage

The resource assessment attribute category contains attributes that describe how resource consumers and allocators observe both the consumption and need for additional resources. These attributes include:

1. Uses for online resource usage feedback: This attribute identifies the resource consumption assessments that are useful for runtime reallocation and tuning. These assessments include performance utility, control error terms, safety margins, etc.

2. Uses for design-time assessment: This attribute identifies resource attributes to be considered during design-time. These assessments include resource complexity, etc.

III. RESOURCE SHARING ONTOLOGY

We have designed an ontology to capture the above attributes and categories of attributes of a shared resource language. We followed general ontology design principles to make the ontology as simple as possible so that it is as extensible as possible and widely applicable [8].

The ontology incorporates the system entities such as resources, system actors, measurements and performance assessments. These entities and several others are represented as classes in the ontology. The ontology also incorporates object and data properties that represent the attributes and attribute categories we discussed above.

A. Shared Resource Ontology Classes

A schematic of the ontology class hierarchy can be seen in Figure 1.

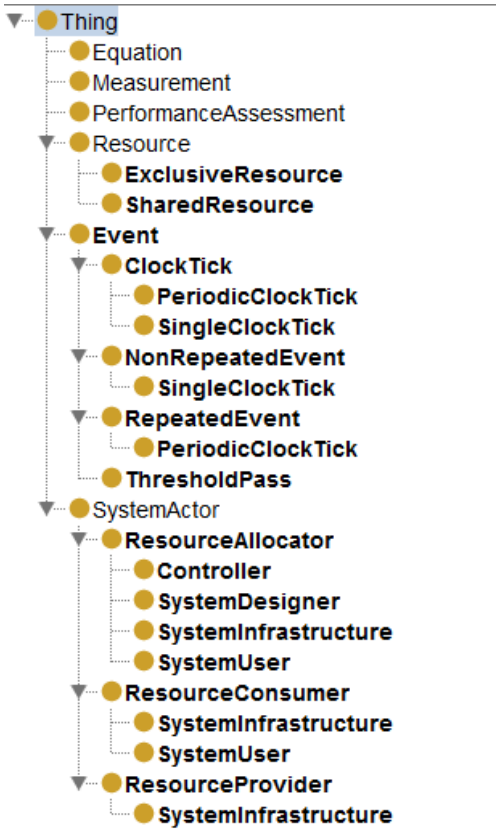

Figure 1: A Tree Representation of the Ontology Class Hierarchy.

There are several main classes of entities in the ontology – Resource, SystemActor, Measurement, PerformanceAssessment, Equation, and Event.

The Resource class represents resources. We define two subclasses – ExclusiveResource and SharedResource to represent resources that are used exclusively by one consumer or multiple consumers, respectively. We expect that shared resources will be prevalent in applications that use this ontology, but that representations of exclusive resources will be necessary for scenarios where some resources are exclusive assigned to safety-critical systems, such as life support in pressurized-cabin air-vehicles.

The SystemActor class represents the system actors. Subclasses include ResourceAllocator, ResourceConsumer and ResourceProvider that represent, respectively, actors that allocate, consume, and provide resources as discussed above. Subclasses of ResourceAllocator include Controller, SystemDesigner, SystemInfrastructure, and SystemUser. Subclasses of ResourceConsumer include SystemInfrastructure and SystemUser. SystemInfrastructure is also a sub-class of ResourceProvider. Controller represents resource allocation controllers such as the engine control module discussed above. SystemDesigner is the system designer and is used when the system designer allocates resources. For example, a designer may decide that some engine designs have a restricted fuel flow to limit power output and increase longevity. SystemUser represents possibly multiple system users which both allocate resources through command decisions and consume resources. For example, the driver of an automobile consumes engine power by accelerating the vehicle. The driver also allocates fuel resources to the engine by adjusting the throttle. SystemInfrastructure represents the system which inherently allocates provides, and consumes resources. For example, modern computer motherboards allocate and provide electrical power to chip components, while also consuming electrical power.

The Measurement, PerformanceAssessment, and Equation classes have no subclasses. Although not introduced earlier, Equation represents mathematical expressions that are used by PerformanceAssessment objects, among others.

The Event class represents discrete points in time that may drive the taking of measurements, running performance assessments, changing allocations and so on. These events may be repeated or not, hence the RepeatedEvent and NonRepeatedEvent classes. Events could occur at clock ticks (repeated or not), or when a measured value or assessment passes some threshold, as is usually the case when using a performance assessment to decide when to perform a resource allocation – when the expected benefit of a reallocation surpasses the expected benefit of not changing a resource allocation, a controller should use this threshold passing to decide to perform a reallocation of resources.

B. Shared Resource Ontology Object Properties

A schematic of the ontology object property hierarchy can be seen in Figure 2.

The identification of the property relationships should be fairly self-explanatory because of the naming conventions we used. As such, we only describe a subset of these properties. Note that the object properties are generally broken into pairs to represent inverse relationships. For example, consumes is used to identify which resources a consumer consumes, while isConsumedBy is used to identify which consumers consume a resource.

A high-level sketch of object property relationship between entities can be seen in Figure 3. To simplify figure 3 and make it more informative and less confusing, we do not show inverse properties whose use can be easily inferred from naming conventions. For example, allocates is an inverse property of isAllocatedBy. We also do not show some subclass entities whose specialized property usage can be easily inferred from naming conventions and the properties of superclasses. For example, exclusivelyConsumes is a subProperty of consumes which is used with ExclusiveResource objects instead of normal Resource objects. We do not have many subProperty relationships in the shared resource object property hierarchy. Most of these object properties are used to identify resource consumptions and availabilities with special exclusivity and observability properties. For example, a monitor connected to a video card frequently exclusively consumes the information flowing from the card. Similarly, in a ground vehicle such as a HMMWV,

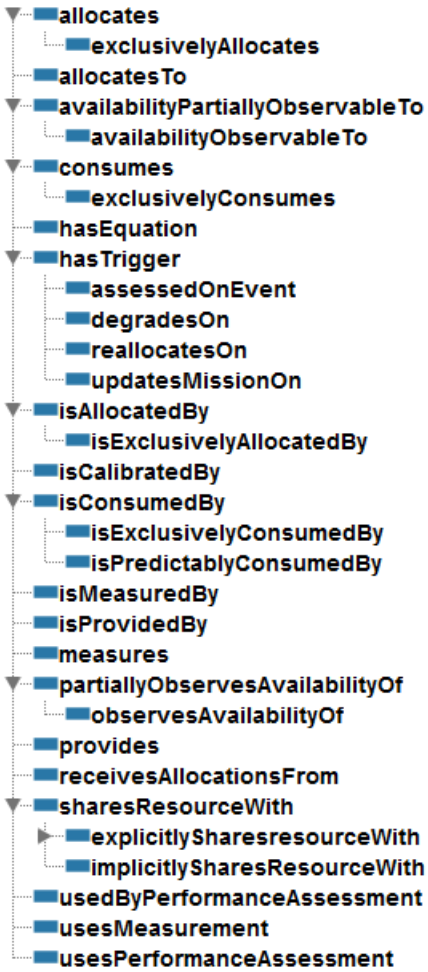

Figure 2: A Tree Representation of the Ontology Object Property Hierarchy.

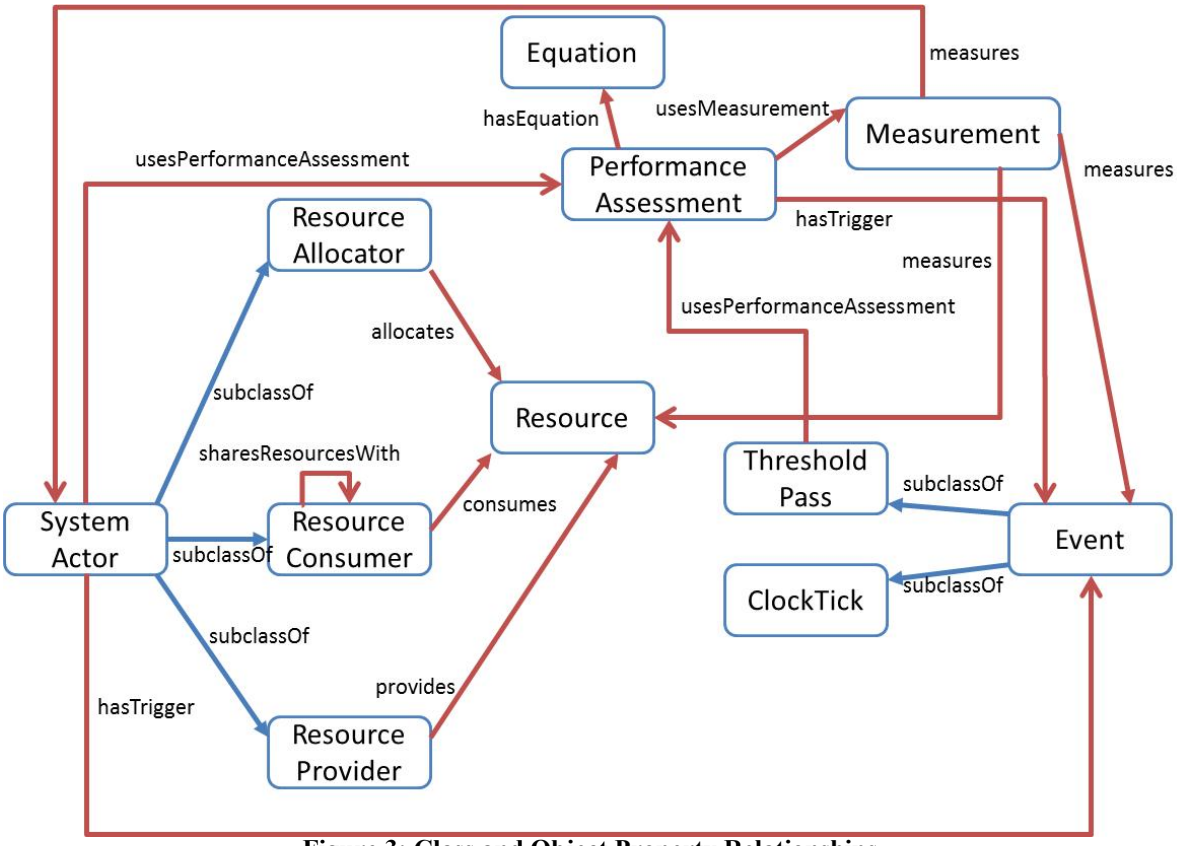

Figure 3: Class and Object Property Relationships.

the engine control unit has the exclusive ability to observe atmospheric pressure in order to regulate the air-fuel mixture into the engine.

We defined the sharesResourceWith property and its subproperties to identify resource consumers that share resources. This sharing may be implicit, explicit or coordinated, so we defined subclasses to capture these scenarios.

We defined hasTrigger properties to identify events which trigger when events are allocated or consumed, or when performance is assessed.

C. *Shared Resource Ontology Data Properties*

A schematic of the ontology data property hierarchy can be seen in Figure 4.

The identification of the property relationships should be fairly self-explanatory because of the naming conventions we used. Most of the data properties represent either times, or they represent strings to describe objects. The time properties include time and period which respectively represent a time when an event occurs and the amount of time between events. The description string properties include resourceDescription, resourceDynamism and most others.

IV. Extending the Shared Resource Ontology for Contention Complexity

As a demonstration of the extensibility of our shared resource ontology, we now discuss an extension of this ontology to cover the assessment of contention complexity metrics.

Contention Complexity is a metric that assesses the propensity for a resource to experience contention. We define each resource to have its own resource complexity measure. We

Figure 4: A Tree Representation of the Ontology Data Property Hierarchy.

codify these and other motivating hypotheses of contention complexity as follows:

1) Entities/subsystems/components in a system use multiple resources. Entities have varying levels of criticality for using specific resources.

2) Contention complexity is a function of the potential for contention due to more requests from entities to use resources than can be accommodated by the limited resources.

3) Contention complexity can be decomposed and expressed for specific resources. Contention complexity is the sum of the contention complexities of resources. The contention complexity of a resource is a function of the potential for contention of that resource from entities to use that resource.

4) Contention complexity of a resource is a function of:

- The number of entities that could request that resource. (A resource with more consumers leads to more contention complexity.)
- The level of usage required for use of the resource. This is measured in terms of % usage level * amount of time per usage. (Higher usage level means more contention, more time per usage means longer queues and more contention.) The higher this product is, the higher likelihood of contention.
- More critical uses of limited resources imply more contention complexity.

This metric is expressed as the following equation:

$$ContentionComplexity(r) = \sum_{c \in DependsOn(r)} \frac{E[\%level(c,r)]var(\%level(c,r))}{criticality(c,r)}$$

We extended the shared resource ontology by defining a subclass of PerformanceAssessment called ContentionComplexity. We then defined three subclasses of Measurement: VariancePercentResourceUsage, ExpectedPercentResourceUsage and Criticality. We defined a new object property forResource that associates ContentionComplexity objects with specific resources. Finally, we define a single individual, ContentionComplexityEquation that expresses the above equation for contention complexity.

With this very small set of changes, we were able to adapt our general shared resource ontology to a much more specific and still useful ontology for a specific metric that can be further refined.

V. Example Applications of Shared Resource Ontology

We now explore the use and usefulness of the resource sharing ontology and its extension to Contention Complexity with two specific examples: an RLC circuit and a hybrid battery maintenance scenario.

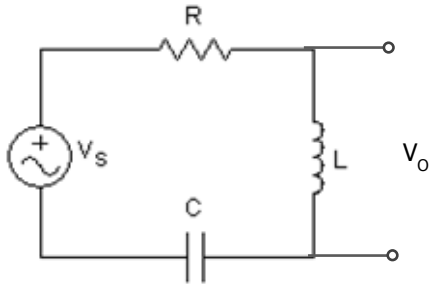

Figure 5: A Tree Representation of the Ontology Data Property Hierarchy.

A. RLC Circuit

We now consider the application of our contention complexity ontology to the RLC circuit in Figure 5.

In this circuit, the voltage source (V_S) is a resource provider and provides a resource of electrical energy. The RLC circuit (without the power source) is both a resource consumer and an allocator. The RLC circuit consumes some energy and determines how much energy at each frequency to transfer to any consumer connected at the V_O output port. Anything that connects to the output port V_O is both a resource consumer and an allocator because it can influence how the power output of V_S is allocated across the frequency domain and hence consumed by the RLC circuit.

We designed an application-specific ontology for this example that imports the contention complexity ontology.

The amount of resource provided at a given frequency s is a function of the voltage $V_S(s)$ and the current through the voltage source $I_S(s)$. This provided electrical energy is allocated by the system designer to the RLC components and an output port which provides a voltage $V_O(s)$ and a variable current $I_O(s)$ that is a function of the systems connected to the output port. We assume that the voltage source has some max power output W_{max}. The power output at a given frequency is $V_O(s)*I_O(s)$ and the integral of this product over the frequencies from 0 to infinity is the total power output.

With this analysis, we designed the application-specific RLC ontology by adding just a few individuals to the imported contention complexity ontology. We added:

- The SourcePowerContention individual which is of type ContentionComplexity. This individual uses measurements of the source voltages and currents, the output voltages and currents, and the RLC circuit parameters to compute contention complexity.
- The SourcePowerOutput individual is of type Resource. It is the resource consumed by the RLC circuit and any other consumer attached to the V_O port.
- The VoltageSource individual is of type Resource Provider. It provides the SourcePowerOutput resource.

- The RLCCircuit and RLCOutput individuals which are of types ResourceAllocator and ResourceConsumer. The RLCCircuit individual has associated data on the RLC values.

- The V_S, I_S, V_O, I_O measurements which are used to computed the resource consumptions.

Based on parameterizations of RLCCircuit, VS, RLCOutput, V_O, or I_O, the RLC-Contention-Complexity ontology can be used to automate the assessment of the contention complexity in the toy circuit.

B. Maintenance Resources for Hybrid Vehicle Battery

We now apply the contention complexity extension of the shared resource ontology to analyze the complexity of maintenance resource sharing over hybrid vehicle lifecycles due to the contention for resources needed to maintain a particular choice of battery. For this application, the more often a battery needs to be replaced, the more resources it will use (e.g., manpower and money) that could be used to maintain other systems. We want to use this metric to select a battery and battery configuration (i.e., the depth of discharge used in the hybrid vehicle's charge/discharge control algorithm) to minimize the maintenance resources needed by the battery over a vehicle's lifecycle for a variety of vehicles based on parameterizations of vehicle weight, battery capacity, the battery charging control, and the vehicle use patterns.

Note that this example is simplified. It does not include other factors that would be involved in the selection of a battery, such as the battery's weight, power density, disposal costs, and safety.

To compute the metric for this example, we need an expected value for the level of demand for the resource (which in this case is the cost to replace the battery) and the variability of that demand. For this example, the depth of discharge has an effect on the demand for the resource. A battery has to be replaced (i.e., demands the resource) after a certain number of charging cycles. A deeper discharge minimizes the number of charging cycles but a shallower discharge enables more cycles before the battery needs to be replaced.

For this scenario, we define two individuals which are of type Resource: PowerStoredInBattery and MaintenanceResources. These individuals respectively capture the ability of the resource to recharge over real or simulated terrains and the requirements for resources for battery maintenance.

There are two individuals of type ResourceProvider that provide the PowerStoredInBattery resources: EnginePower, Battery and RegenerativeBraking. Because maintenance resources are part of the nominal system infrastructure and must be provided for vehicle use, we consider maintenance to be provided finite and provided by the system infrastructure for the purposes of our analysis. Larger analysis might include multiple vehicles of various types to analyze maintenance requirements.

There are two individuals of ResourceConsumer type: Battery and VehicleDrive. The battery has multiple properties associated with it including the cycle capacity for a given depth of discharge which is represented as a PerformanceAssessment individual. Each battery is associated with BatteryType and PowerCapacity properties. Vehicle drive is parameterized with performance requirements.

The BatteryController individual is of type ResourceAllocator and allocates power for either the discharging or recharging of the battery.

There are multiple events that are used to control the allocation of resources based on whether the engine turns on or off, whether the vehicle is drawing power from the battery, and whether the vehicle needs motive power or not. These measurements are used by the battery charger to decide when to charge the battery. The control operation is ultimately measured by a Measurement individual for the battery: the number of recharging cycles experienced by the battery.

The battery control algorithm is based on a straightforward Energy Transfer Model. Basically, going up a hill at a particular speed uses a certain amount of energy (potentially provided by discharging the battery) and going down a hill transforms potential energy into kinetic energy that can be stored in the battery as electro-chemical energy. We designed our simple control algorithm so that the battery will be used for motive power until it is fully discharged (to its prescribed depth of discharge), then it would charge until it is fully charged – either by the engine or regenerative charging system.

Our intention is that this ontology we are sketching for hybrid vehicle battery maintenance is used to analyze the maintenance requirements for individual vehicles or fleets of vehicles. We see it being used to compute the lifecycle replacement cost for batteries' replacement. We used an early version of this ontology to run simulations for multiple batteries at 10 different depth of discharge levels (10% through 100%), with each simulation run covering 10,000 hours of vehicle operation, and kept track of how many charging cycles occurred over the 10,000 hours of operation. The graph in Figure 6 shows preliminary results of the computed metric graphed for each battery and depth of discharge on a logarithmic scale.

We should be careful to note that as currently designed, assessments using the ontology will always admit some error because, for reasons of tractability and abstraction to simplify analysis, the ontology does not include all the factors that should go into the battery choice. For example, our approach does not account for safety concerns. (Li-ion batteries are

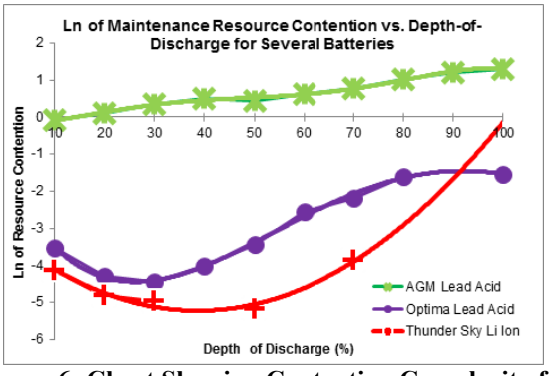

Figure 6: Chart Showing Contention Complexity for Multiple Battery Types and Depths of Discharge.

known to be explosive when engulfed in flames.) But in this simplified example, the computation of the metric would show that the choice of the Lithium Ion battery with a depth of discharge between 30% and 50% provides the minimal resource contention complexity (informally, the minimal maintenance cost over the battery's lifetime). Interestingly to us is that these preliminary results align well with real-world results that we discovered from automakers after performing these experiments.

VI. CONCLUSION

In this paper we introduce a resource sharing ontology to address the challenge of modeling and reasoning about explicit and implicit resource interactions for integrated systems and services. The resource sharing ontology is driven by the identification of various classes of system actors, resource attributes, measurements and performance assessments. Using a contention complexity metric we discuss the extensibility of the ontology, and we discuss applications to examples.

REFERENCES

[1] L. Alberts. YMIR: an ontology for engineering design. Doctoral dissertation, University of Twente, 1993.

[2] P. Chandra, A. Fisher and P. Steenkiste. A Signaling Protocol for Structured Rresource Allocation. In: INFOCOM '99. Eighteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE. Vol. 2, pgs 522-533, 1999.

[3] T. Gruber and G. Olsen. An ontology for engineering mathematics. In J. Doyle, P. Torasso, & E. Sandewall (Eds.), Fourth International Conference on Principles of Knowledge Representation and Reasoning, Gustav Stresemann Institut, Bonn, Germany, Morgan Kaufmann. A well designed ontology, built to enable sharing and reuse of engineering models. 1994.

[4] K. Krauter and R. Buyya and M. Maheswaran. A Taxonomy and Survey of Grid Resource Management Systems. Software Practice and Experience 32(2):135-164, 2002.

[5] S. Lemaignan, A. Siadat, J. Dantan and A. Semenenko, "MASON: A Proposal For An Ontology Of Manufacturing Domain," Distributed Intelligent Systems: Collective Intelligence and Its Applications, IEEE Workshop on, pp. 195-200, IEEE Workshop on Distributed Intelligent Systems: Collective Intelligence and Its Applications (DIS'06), 2006 H.-K. Lin, J. Harding and M. Shahbaz. "Manufacturing system engineering ontology for semantic interoperability across extended project teams" International Journal of Production Research 42.24 (2004). 19 May. 2011

[6] M. Maheswaran. Quality of service driven resource management algorithms for network computing. 1999 Int'l Conference on Parallel and Distributed Processing Technologies and Applications (PDPTA '99), June 1999, pp. 1090-- 1096.

[7] M. Obitko and V. Marik. Ontologies for multi-agent systems in manufacturing domain. In DEXA '02: Proceedings of the 13th International Workshop on Database and Expert Systems Applications, pages 597–602, Washington, DC, USA, 2002. IEEE Computer Society.

[8] OWL. (2011) Web Ontology Language (OWL.) Retrieved from http://www.w3.org/TR/owl2-overview/

[9] L. Pouchard, N. Ivezic and C. Schlenoff (2000) "Ontology Engineering for Distributed Collaboration in Manufacturing". In Proceedings of the AIS2000 conference, March 2000.

[10] S. Ray. "Tackling the Semantic Interoperability of Modern Manufacturing Systems" Proceedings of the Second Semantic Technologies for eGov Conference (2004)

[11] A. Rector, G. Schreiber, N. Noy, H. Knublauch and M. Musen. Ontology Design Patterns and Problems: Practical Ontology Engineering using Protege-OWL. Tutorial at the Third International Semantic Web Conference (ISWC 2004) November 7th, 2004

[12] K. Rohloff, R. Schantz and Y. Gabay. "High-Level Dynamic Resource Management for Distributed, Real-Time Embedded Systems." Submitted to the 5th Symposium on Design, Analysis and Simulation of Distributed Systems, 2007.

[13] K. Rohloff, J. Ye, J. Loyall and R. Schantz. A Hierarchical Control System for Dynamic Resource Management, 2006 IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS 2006), Work in Progress Symposium. April 7, 2006, San Jose, CA.

[14] K. Rohloff, J. Loyall, R. Schantz. Quality Measures for Embedded Systems and Their Application to Control and Certification, ACM SIGBED Review, Special Issues on Workshop on Innovative Techniques for Certification of Embedded Systems. Volume 3, Number 4, October 2006.

[15] C. Schlenoff, R. Ivester, and A. Knutilla. A robust process ontology for manufacturing systems integration. In Proceedings of 2nd International Conference on Engineering Design and Automation, 1998.