Towards an Artificial Space Object Taxonomy

Matthew P. Wilkins
Applied Defense Solutions, Columbia, MD
Avi Pfeffer
Charles River Analytics, Cambridge, MA
Paul W. Schumacher
Air Force Research Laboratory, Kihei, HI
Moriba K. Jah
Air Force Research Laboratory, Albuquerque, NM

1. ABSTRACT

Object recognition is the first step in positively identifying a resident space object (RSO), i.e. assigning an RSO to a category such as GPS satellite or space debris. Object identification is the process of deciding that two RSOs are in fact one and the same. Provided we have appropriately defined a satellite taxonomy that allows us to place a given RSO into a particular class of object without any ambiguity, one can assess the probability of assignment to a particular class by determining how well the object satisfies the unique criteria of belonging to that class. Ultimately, tree-based taxonomies delineate unique signatures by defining the minimum amount of information required to positively identify a RSO. Therefore, taxonomic trees can be used to depict hypotheses in a Bayesian object recognition and identification process. This work describes a new RSO taxonomy along with specific reasoning behind the choice of groupings. We will demonstrate how to implement this taxonomy in Figaro, an open source probabilistic programming language.

2. INTRODUCTION

Currently, US Space Command routinely tracks on the order of 20,000 to 30,000 objects. However, some estimates place the true size of the space object population in Earth orbit upwards of 100,000 to 500,000 objects. Considering the fact that space objects greater than 1 *cm* in diameter can produce catastrophic collision energies and the most minute debris particles can produce damaging effects over time, it is imperative that the astrodynamics and sensor communities work together to provide a complete catalog of space objects. The bulk of low earth orbit (LEO) RSO tracking and cataloging comes from the Space Fence, providing range and range rate data; however, routine tracking of objects less than 10 *cm* in diameter is difficult due to radar sensor physics. The planned S-Band radar fence to replace the current Space Fence is predicted to be able to track a large number of newly detectable small and/or dim objects not currently cataloged. For geosynchronous (GEO) object sensing, we rely on electro-optical telescopes where small, distant objects are faint and yield weak signal-to-noise ratios for radiometric detection. This leads to sparse angles-based observations and large uncertainties in estimated quantities, yielding false cues with standard estimation techniques. Furthermore, clustered space objects can appear to cross paths with other space objects often and are frequently mis-identified and mis-tagged.

Even if we could detect and track the smallest of debris particles, the current cataloging systems cannot properly associate all of the data required to maintain the orbits of 100,000 objects. The current method of track custody for just 20,000 objects involves the "lost list." This list contains items detected and/or tracked at one time but we cannot accurately predict where these objects are to properly cue new follow-up kinematic and radiometric measurements. As a result, when we identify a "new" space object, we must determine whether the "new" object is truly a new object (not seen before) or, more likely in the current cataloging system, a "lost" object.

The data association problem, which is inherently difficult, is compounded by the difficulty of dealing with errors in complex dynamics and feature discrimination. The process, sensor, and model error distributions associated with target tracking and cataloging are not properly preserved by traditional methodologies. Furthermore, these errors all combine in such a way that we cannot make the data association determination with any significant degree of confidence using traditional methods. Underlying all of this are the highly complex dynamics and the ensuing highly complex models that attempt to capture or represent the complex dynamics. In particular, dynamics of space objects are modeled with a combination of discrete and continuous processes that are difficult to reason with. Existing approaches, such as the wide variety of filters available for orbit determination, work best for specific types of

objects and do not always provide a holistic view. They also do not generally deal with data association and require a separate model and/or methodology for this problem.

Our vision is to move away from human experiential decision making based upon orbit error ellipsoids and move towards an automated agent that can assemble sparse and disparate information sources into actionable knowledge to support better decision making. We envision a methodology that can not only perform traditional catalog queries such as "*What is in my general vicinity?*" but also can use experiential evidence and historical trends to answer questions such as *"Should I take steps to protect my satellite?"*

To accomplish this, we need a holistic framework for representing and reasoning about the dynamics, associated errors, and characterization of space objects. The first step of this process is to construct an appearance model for resident space objects in the form of a space object taxonomy. This taxonomy will become the basis for the holistic framework needed to reason algorithmically about the characterization of space objects, generate hypotheses for object recognition and identification, and to perform data association.

This work describes a new RSO taxonomy along with specific reasoning behind the choice of groupings. Our purpose here is twofold. First, we aim to continue the conversation about the necessary and sufficient structure of a RSO taxonomy that was begun by Früh et al. [1] Second, we want to emphasize the usefulness of having a robust RSO taxonomy. Modern biological taxonomy has been a work in progress for well over the last century, and we fully expect that ours will be an evolving conversation in the literature.

In addition to describing our proposed taxonomy, we will also demonstrate how to implement the taxonomy in Figaro™, an open source probabilistic programming language developed by Charles River Analytics. State of the art probabilistic programming has been applied to open universe situations where you do not know what objects exist, how many exist, or their relationships. [2] [3] This flexibility is ideal for our needs in the SSA domain. The Figaro software is freely available for download, and comes with an extensible library of validated reasoning algorithms and a detailed tutorial. [4] [5]

Figaro is extremely expressive and can represent a wide variety of models, including directed and undirected models, models in which conditions and constraints are expressed by arbitrary Scala functions, models involving inter-related objects, open universe models in which we do not know what or how many objects exist, models involving discrete and continuous elements, models in which the elements are rich data structures, such as trees. Figaro provides a rich library of constructs to build these models, and provides ways to extend this library to create your own model elements.

Figaro's library of reasoning algorithms is also extensible. Current built-in algorithms include exact inference using variable elimination, importance sampling, Metropolis-Hastings sampling with an expressive language to define proposal distributions, support computation, and most probable explanation (MPE) using variable elimination. Figaro provides both regular and anytime (i.e. interrupt your algorithms at any step to inject new evidence or conditions) reasoning algorithms. In addition to the built-in algorithms, Figaro provides a number of tools for creating your own reasoning algorithms. Figaro can also represent dynamic models and provides a particle filtering algorithm for such models.

3. TAXONOMY OVERVIEW

The best example of a taxonomy is the classical Linnaean biological taxonomy because it has almost universal scientific acceptance. A strength of Linnaean taxonomy is that it can be used to organize the different kinds of living organisms, simply and practically. Every species can be given a unique name. This uniqueness and stability are a result of the acceptance by biologists specializing in taxonomy, not merely of the binomial names themselves. While there is no specific rule for creating individual *taxa* (i.e. a grouping), the taxonomy is governed by rules for naming these taxa, which are laid down in formal Nomenclature Codes. We seek to provide a similar system through a defined tree-based RSO taxonomy structure.

The structure of the Linnaean Taxonomy, as illustrated in Figure 1, makes it a relatively straightforward job to derive a corollary for satellites. For a biological taxonomy, species can be placed in a ranked hierarchy, starting with either *domains*, which are divided into Kingdoms. Kingdoms are divided into *phyla*. Phyla are divided into *classes*, and they, in turn, into *orders*, *families*, *genera*, and *species*. A unique path, or signature, through the tree based taxonomy describes any specific species.

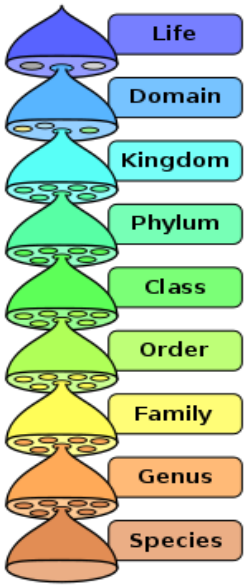

Figure 1: Linnaean Taxonomy (from Wikimedia Commons)

As a corollary for space objects, Domain would represent either Natural or Artificial Satellites. Kingdom represents the primary celestial body at the focus of the orbit. Phyla would represent particular celestial body orbit regimes. Classes would begin the distinction of mission types (Navigation, Communication, Meteorology) and body plans (e.g. box-wing, cubesat, solar sail). Whatever taxonomy is initially decided upon, it will by necessity change over time. For example, solar sail technology will represent a new and never-before-seen form factor for spacecraft and the taxonomy will need to accommodate this change. Ultimately, the taxonomy is a library or catalog of spacecraft models and properties. The taxonomy allows one to determine the most prominent features of a spacecraft first, before delving into lower tree levels to ascertain more obscure or detailed properties.

4. TAXONOMIC TREES DEPICT HYPOTHESES

Now consider the usefulness of having a robust RSO taxonomy. Each categorization within a taxonomy, beginning with the most general or inclusive, at any level, is called a *taxon* (See Figure 2). As an example, Figure 3 is a representation of a biological tree structure where the unique signature of *Catharanthus roseus* (Madagascan Periwinkle) is displayed. Provided that we have appropriately defined a satellite taxonomy that allows us to place a given object into a particular *taxon* without any ambiguity, one can assess the probability of assignment to a particular class by determining how well the object satisfies the unique criteria of belonging to that class. From this we can say that o*bject recognition* is the first step in positively identifying a resident space object (RSO), i.e. assigning an RSO to a category such as GPS satellite or space debris. Beyond assigning an object to a very specific "species" or "sub-species" taxon, o*bject identification* is the process of deciding that two RSOs are in fact one and the same. Ultimately, tree-based taxonomies delineate unique signatures by defining the minimum amount of information required to positively identify a RSO.

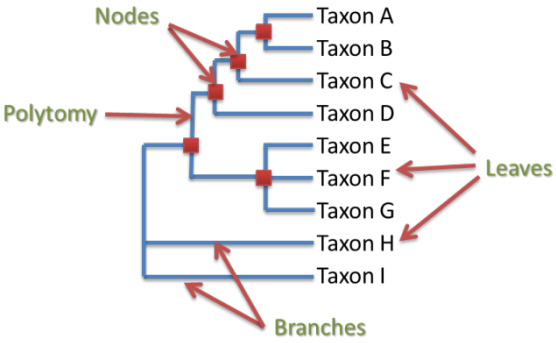

Figure 3: Taxonomic Trees

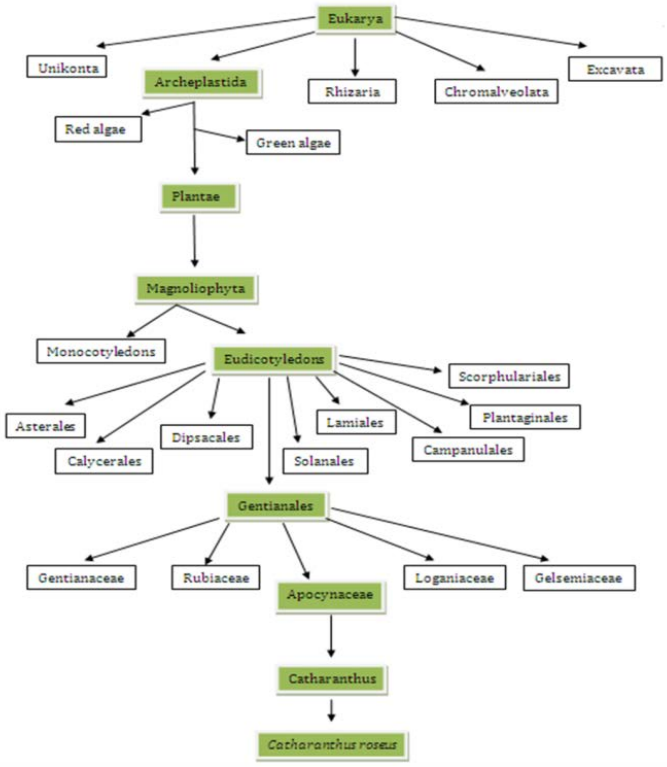

Figure 2: A taxonomic tree allows one to construct a unique signature.

Therefore, taxonomic trees can be used to depict and reason about hypotheses in a Bayesian object recognition and identification process.

Each taxon will have a set of uniquely distinguishing characteristics that will allow one to place a given object into a specific group without any ambiguity. When a new object does not fall into a previously defined taxon, the entire tree structure will need to be evaluated to determine if a new taxon should be created. Ultimately, an automated learning process to facilitate tree growth would be desirable. The key concept is that one can assess the probability of assignment to a particular taxon by determining how well the object satisfies the unique criteria of belonging to that taxon combined with the probability of belonging to each of the ancestral nodes. Therefore, we can use taxonomic trees in a Bayesian process to assign prior probabilities to each of our object recognition and identification hypotheses.

5. CHARACTERISTICS OF A RSO TAXONOMY

Following the principle that taxonomic distinctions must be made in such a way that one can place an object into a group without ambiguity, the distinguishing characteristics of the highest and most inclusive groupings of resident space objects are: Natural (Naturally occurring objects) and Artificial (Man-made objects). This level of distinction is equivalent to the biological Domain. The astronomy community has already developed a taxonomy for classifying the various naturally occurring celestial bodies such as stars, planets, and other astronomical phenomena. We will not discuss this natural-object taxonomy in detail here; however, there are a variety of lessons to be learned by studying this taxonomy along with the Linnaean taxonomy to aid in further dividing artificial objects into more specific taxa. [6] [7]

Even a cursory glance at the modern Linnaean taxonomy for living organisms will reveal that one of the most basic distinctions comes from the method of observation. From the time of Aristotle, living things were divided into Plants and Animals. However, the advent of microscopy and the ensuing electron-microscopy forced taxonomists to create new Kingdoms called Fungi, Protista, and Monera. In the natural celestial-object taxonomy, one need only look at Binary Stars, which are divided into five groups: Eclipsing, Visual, Astrometric, Photometric, and Exotic. Each of these classifications relies exclusively upon the method of observation. Creating taxa based upon the method of observation has not only proven to be a workable classification system but also has the added benefit of being able to include or exclude large sections of the taxonomical tree structure for the purpose of object identification based upon the observation data source alone.

From this we can say that detectability of an RSO is one of the most important and uniquely identifying distinctions that can be made at this level. One will never confuse an object orbiting the sun with an object orbiting the Earth simply by virtue of the fact that we must use very specific sensors and techniques to detect, track, and identify objects that are far way from the Earth's surface. Therefore, for artificial object, we propose the most basic distinction is the central body about which an artificial object is orbiting. To that end, the artificial object (AO) "Kingdoms" are broken down by central body: Sun orbiting, Mercury orbiting, Venus orbiting, Earth orbiting, Mars orbiting, etc. In each class, a list of sensors that can detect objects in that regime can provide simple distinction.

Now that the central body has been established, we find that the next most basic distinction is the orbit regime about the central body, which comprises the RSO "Phyla." We can see this distinction from two perspectives: dynamical models and, once again, sensors. Astrodynamicists know from theory and experiment that different orbit regimes require knowledge and understanding of a variety of conservative and non-conservative forces. From a dynamic model point of view, the theories required to properly model the motion of a Low Earth Orbit (LEO) object are distinct from Geostationary Earth Orbit (GEO) theories. In general, specific orbit theories have different choices of state representation and include or exclude various non-conservative force models among other differences. From a sensor point of view, we again argue that one will never confuse a LEO object with an object at geosynchronous altitude because we generally use different classes of sensors and techniques to detect, track and identify objects in each orbit regime. The mere fact that an object is detectable by a certain sensor provides you with a wealth of information that should be reflected in the taxonomical structure, and this distinction can be made prior to ever performing orbit determination on the object itself. We note that physical characteristics of RSOs such as shape models and area to mass ratio are still *not* a consideration at this point.

Therefore, RSO "Phyla" have the following distinctions. First, bounds can be placed on the orbit elements that are tantamount to the classical apogee-perigee filter from conjunction analysis. Second, one can list specific sensors that are capable of detecting RSOs in a particular orbit regime. Third, the non-conservative forces in each regime and their required dynamical models to perform accurate orbit determination can be used as a distinction such as

atmospheric effects, gravitational perturbations, and solar radiation pressure. In the bulleted list below, we have attempted to enumerate these distinctions and broaden these classifications to describe an arbitrary central body. Note that these feature sets can be tailored to the Earth and using the standard LEO, MEO, GEO, HEO terminology when referring to the Earth Orbiting classification.

Low Central Body Orbit (LCBO)

- Has strong atmospheric effects (if atmosphere is present)
- Has strong gravitational perturbations
- Apoapsis and periapsis both exist in the same region
- Apoapsis is below the lowest circular orbit altitude of the MEO regime
- Periapsis is above the lowest circular orbit altitude before the object naturally de-orbits due to atmospheric effects.
- For Earth, Solar Radiation Pressure perturbations are much less significant than drag or gravity effects
- Altitude less than 2000 km for Earth

Medium Central Body Orbit (MCBO)

- Has mild atmospheric effects
- Has gravitational perturbations
- Apoapsis and periapsis both exist in the same region
- Periapsis is above the highest circular orbit altitude of the LCBO regime
- Apoapsis is below the lowest circular orbit altitude of the CBSO/HCBO regime
- For Earth, SRP and drag forces may be on the same order of magnitude
- Altitude between 2000 and 35876 km for Earth

High Central Body Orbit (HCBO)

- Apoapsis and periapsis both exist in the same region
- Periapsis is above the highest circular orbit altitude of the MCBO regime
- Apoapsis is any altitude that is still within the central body gravitational sphere of influence
- Primary orbit perturbations are third-body gravitational effects.
- Altitude above 35786 km for Earth

Central Body Stationary Orbit (CBSO)

- Has orbit period that matches central body rotational period about its spin axis.
- Altitude nearly 35876 km for earth

Highly Eccentric Orbit (HEO)

- Periapsis and apoapsis exist in two different orbit regimes
- Because these objects are crossing through various orbit regimes, they can be confused at any given instant with objects that never leave that particular regime
- The orbit eccentricity is a primary distinguishing factor
- Dynamical models must accommodate multiple flight regimes

On the next level of distinction down the taxonomical tree, called artificial object “Class” where all RSOs now exist at similar orbit altitudes and generally do not cross into other orbit regimes (except for HEOs), we now must look at object mission profiles. Mission profiles govern the overall design of a satellite. The following list is a basic set of distinguishing characteristics for a given mission that is not meant to be all inclusive. Other feature sets may be

required as work progresses on this effort. Note that an object with no discernible mission would be classified as a defunct payload, rocket body, or debris fragment at this level.

- Area to Mass ratio ranges
- Oblate vs prolate shape
- Active or passive sensors?
- Active broadcaster antennas, passive collector antennas, or both?
- Has extended solar panels or confined to body shape?
- Has a controlled attitude and/or spin state?
- Has a controlled orbit?

Here we skip the traditional biological classification of "Order" intentionally because we feel that the term "Family" is more *apropos*. This level of distinction is governed by the manufacturers of the spacecraft. We assume that particular contractors will have a small set of common satellite bus structures that will be used to design for future missions. Given the risk-averse space industry that relies upon proven technologies, it is not unreasonable to assume that manufacturers are chosen to design a spacecraft for particular missions because of their past performance in particular mission areas. Given the expense and limitations of launching a satellite, we further assume that spacecraft designs are parsimonious: a satellite will never have more physical characteristics than what a mission calls for (i.e. why have 3 solar panels when 1 will do?).

- Specific Area to Mass ratios
- Number, size, and type of sensors
- Number and size of antennas
- Number, size, placement, and orientation of solar panels
- Type of attitude control (CMGs, torque rods, reaction wheels, etc)
- Type of orbit control (finite burn thrusters, constant thrust, etc)
- Type of spin control
- Generic spacecraft bus size and shape

Objects that are defunct payloads and rocket bodies fall into an "inactive" mission category. However, they would still be classifiable by known manufacturer information. Debris fragments, on the other hand, require a different strategy for distinction. We refer the reader to Früh et al. for a more focused look at grouping debris fragments, which is a classification containing the largest number of objects. [1]

For active objects, the distinction of Genus and Species begin to express the distinctiveness of particular satellite busses, number and size of antennas, as well as number and size of solar panels. We are proposing to use the satellite bus classification work of Tamara Payne et al. as a starting point for this level of classification. [8] [9] [10] [11] [12] Beyond this point, the taxonomical tree requires more details than we have space to discuss. We remind the reader that there are entire books devoted to the detailed biological classification schema.

6. RSO TAXONOMY FOR ARTIFICIAL SPACE OBJECTS

Now that we have described the process of how we propose to construct a satellite taxonomy, here we present a graphical representation of the RSO taxonomy for artificial objects as we have currently defined it (See Figure 4). At the rank of Family, the mission structure starts to become unwieldy for graphical representation. Figure 5 presents the tree-based taxonomy from the rank of Class through Family in a more tabular form. For the mission types, we have chosen to adopt the nomenclature used by the Union of Concerned Scientists (UCS) in their UCS Satellite Database, which is freely distributed via their website. [13] The UCS Satellite Database is a listing of all currently active artificial satellites and includes basic information about their launch parameters, mission profile but does not contain detailed information necessary to locate individual satellites.

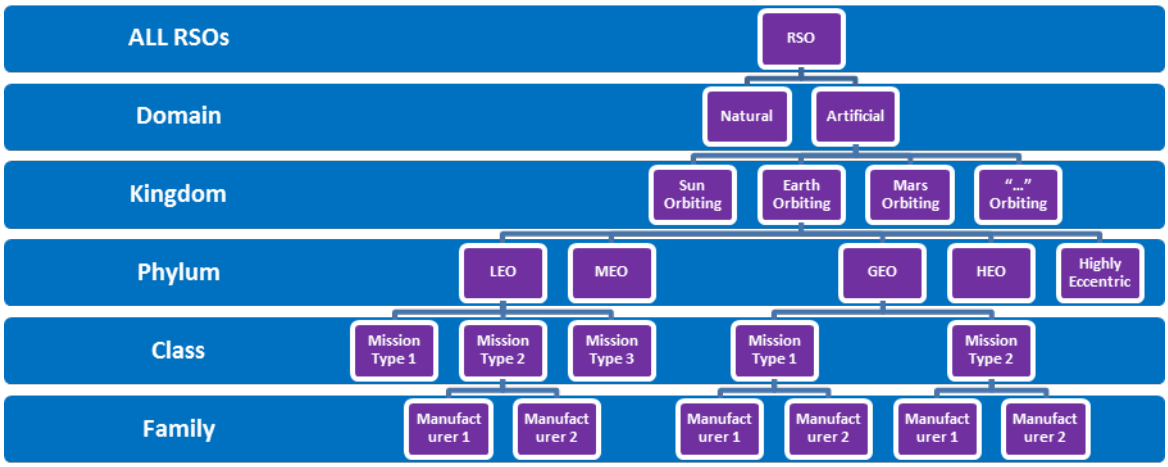

Figure 4: RSO Taxonomy for Artificial Objects

LEO
- Communications
  - Amateur Radio
  - Commercial Broadcast
- Electronic Surveillance
- Navigation
- Earth Observation
  - Surveillance
  - Reconnaissance
  - Meteorology
  - Oceanography
  - Maritime Tracking
- Inactive
  - R/B
  - Defunct Payloads
  - Debris
- Space Observation
  - Solar Physics
  - Space Physics
  - Surveillance

MEO
- Navigation
- Inactive
  - R/B
  - Defunct Payloads
  - Debris

GEO
- Inactive
  - R/B
  - Defunct Payloads
  - Debris
- Communications
- Early Warning
- Earth Observation
  - Meteorology
- Electronic Surveillance
- Navigation
- Space Observation
  - Space Physics

HEO
- Space Observation
  - Space Physics

Highly Eccentric
- Inactive
  - R/B
  - Defunct Payloads
  - Debris
- Communications
- Early Warning
- Electronic Surveillance
- Space Observation
  - Space Physics

Figure 5: RSO Taxonomy for Artificial Objects continued.

7. CONSTRUCTING AN ALGORITHMIC SATELLITE TAXONOMY

Constructing an algorithmic representation of a RSO taxonomy requires some excursion into the realm of computer science. There are two major tree representation strategies: Binary Trees and General Trees. A binary tree, depicted in Figure 6, consists of a node called the "root" together with two binary trees called the left sub-tree and the right sub-tree of the root, which are disjoint from each other and from the root. Any given binary node has at most two children but can also have zero or one child. A general tree, depicted in Figure 7, consists of a root node that has an arbitrary number of child nodes, and each child node can have an arbitrary number of children as well. The children

of a particular node are collectively called siblings. Binary tree implementations are preferred because they have distinct advantages when it comes to searching the tree for a particular node. However, our RSO taxonomy is most easily expressed and visualized as a General Tree.

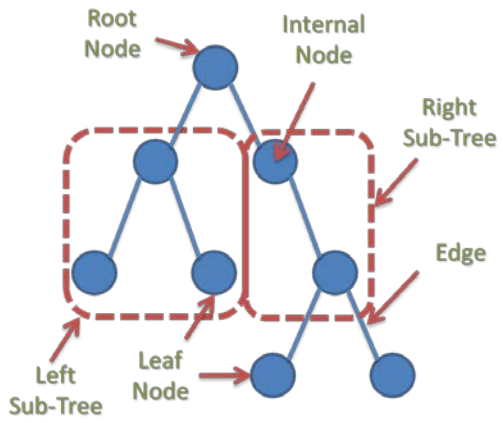

Figure 6: Binary Tree Representation

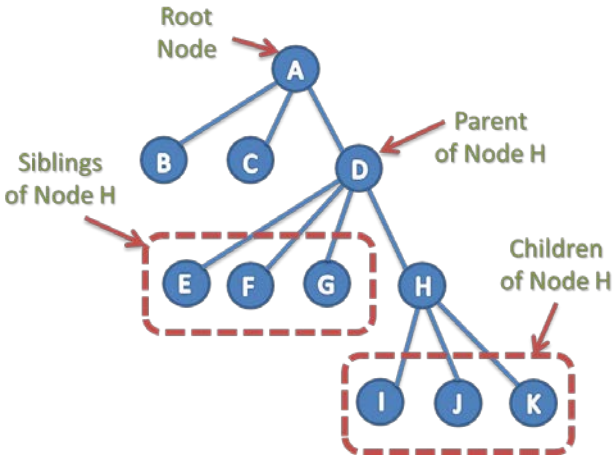

Figure 7: General Tree Representation

Fortunately, there is a one-to-one mapping between general trees and binary trees. To convert a general tree to binary tree, we need only to think of the binary left and right sub-trees as the general tree left child and right sibling. Each node *N* in the general tree corresponds to a node *N'* in the binary tree; the *left* child of *N'* is the node corresponding to the first child of *N*, and the *right* child of *N'* is the node corresponding to *N*'s next sibling. This binary tree representation of a general order tree is sometimes also referred to as a left child-right sibling binary tree (LCRS tree), or a doubly chained tree, or a Filial-Heir chain. Figure 8 graphically depicts a general tree that has been mapped into a LCRS binary tree. While the general tree structure is needed to represent ancestral relationships such as the unique path from the root node to a leaf node, having the capability to map the general tree to a LCRS binary tree allows us to take advantage of existing binary search tree strategies that will significantly speed up computations when we need to represent 100,000 RSOs using a tree structure.

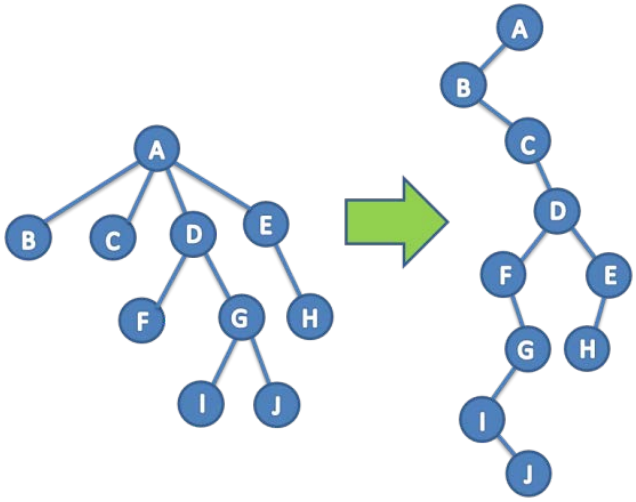

Figure 8: Converting a General Tree to a LCRS Binary Tree

8. PROBABILISTIC PROGRAMMING

The model domain of space object detection, tracking, identification and characterization is complex. In an attempt to accurately represent or simulate observational data, researchers have developed models for everything from the approximate shape of a satellite, the conservative and non-conservative forces acting on the satellite, to the sensor models emulating what kind of data they detect. It is possible to create models of varying degrees of fidelity to represent each of these aspects of the domain, but such a piecemeal approach is extremely expensive. Probabilistic programming offers a better solution because it provides a general language to represent models and built-in reasoning algorithms that apply to all models represented in the language. Furthermore, probabilistic programming provides the means to combine different aspects of the domain in a coherent, holistic model, without having to develop new representations for the combinations.

Probabilistic programming works by using programming language theory to structure probabilistic models and their reasoning algorithms. [14] [15] [16] Charles River's probabilistic programming language, Figaro, is founded on functional programming concepts. [4] In functional programming, an expression defines a computation, producing a value; in Figaro, a model element describes a computation that *stochastically* produces a value. The meaning of the element is the probability distribution over the value produced by the computation. One can then use Figaro's reasoning algorithms to reason about this distribution, for example, to compute the conditional probability of some aspect of the output given observations.

Figaro provides the standard features of functional programming languages, such as variables, functions, conditional execution, recursion, and data structures. Therefore, it enables representation of models with rich data structures and complex control flows. Any aspect of the model domain that can be described by a program can be represented. In addition, Figaro provides the ability to define constraints on language elements, further increasing the representational power.

What Exactly is Figaro?

To be precise, Figaro is a Scala library. It defines rich data structures for probabilistic models and reasoning algorithms for reasoning with those models. Because these are Scala data structures, Figaro models can be created using the full power of Scala. These three things are the key to Figaro:

1) The ability to represent an extremely large and interesting class of probabilistic models using these data structures.
2) The ability to use a reasoning algorithm on these data structures to draw conclusions about the probabilistic model.
3) The ability to create and manipulate the data structures using Scala.

Figaro is also extensible, making it easy to create new kinds of data structures in the library, and, while developing new algorithms is a more complex task, Figaro also provides the means to develop new algorithms for the library. Figaro is extremely expressive by virtue of the fact that it can represent a wide variety of models, including:

- directed and undirected models
- models in which conditions and constraints are expressed by arbitrary Scala functions
- models involving inter-related objects
- "open universe" models in which we don't know what or how many objects exist
- models involving discrete and continuous elements
- models in which the elements are rich data structures such as trees

Figaro provides a rich library of constructs to build these models, and provides ways to extend this library to create your own model elements.

Figaro's library of reasoning algorithms is also extensible. Current built-in algorithms include:

- Exact inference using variable elimination
- Importance sampling
- Metropolis-Hastings, with an expressive language to define proposal distributions
- Support computation

- Most probable explanation (MPE) using variable elimination

Figaro provides both regular and anytime versions of some of these algorithms. In addition to the built-in algorithms, Figaro provides a number of tools for creating your own reasoning algorithms. Figaro can also represent dynamic models and provides a particle filtering algorithm for such models.

9. EXAMPLE SIMPLE TAXONOMY AND FIGARO REPRESENTATION

Here we present an example of a simplified RSO Taxonomy to demonstrate how one would implement the taxonomy using Figaro. In this model, all RSOs are broken up into three categories: payloads, rocket bodies, and debris. Payloads can either be sun pointing, nadir pointing, spin stabilized or uncontrolled. Rocket bodies and debris fragments are both uncontrolled objects. If a payload has controlled pointing, it is considered an active object. Otherwise, an uncontrolled payload along with rocket bodies and debris fragments are considered inactive. Furthermore, payloads and rocket bodies are considered to be intact, low area-to-mass ratio objects while debris objects are considered fragments with a range of sizes from low to high area-to-mass ratio.

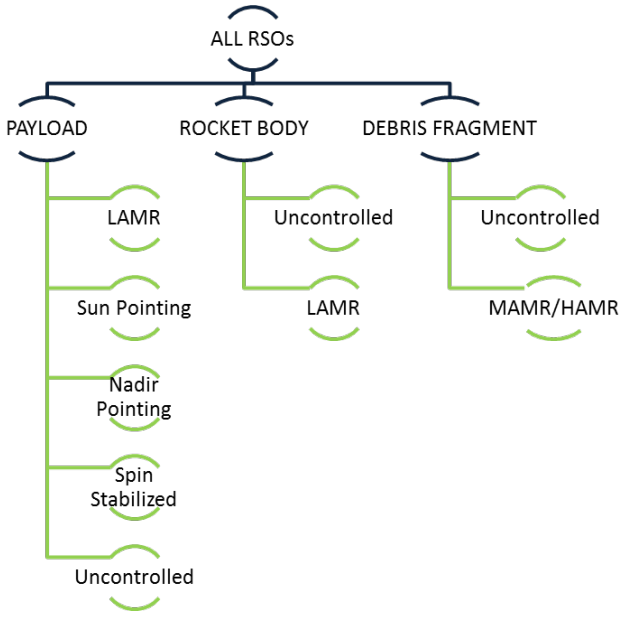

Figure 9: Basic RSO Taxonomy (top level) with characteristics of each class listed below.

From this model, there are three basic characteristics to consider that each have discrete values: area-to-mass ratio, attitude control, and spin state. Each of the object types (payloads, rocket bodies, and debris fragments) will have a different distribution associated with these characteristics. Table 1 has the allowable discrete values for each characteristic along with the initial *a priori* distribution. The *a priori* distribution presented here is extremely general for example purposes only.

We can then write each of the characteristics and distributions in Scala using Figaro elements. An element is the fundamental Figaro data structure, representing a random variable that takes on values in some domain. Elements are parameterized by the type of value they take; for example, an Element[Symbol] is an element that takes values of type Symbol. In the following code example, we use three types of element classes. `Constant('low)` represents the random variable that always takes on the symbol 'low. `Uniform(value`$_1$`,…,value`$_n$`)` represents the discrete uniform distribution over the given values. `Select(0.4 -> "a", 0.6 -> "b")`is an Element[String] that represents the probabilistic model that produces "a" with probability 0.4 and "b" with probability 0.6. For more detailed information about the various probabilistic models contained by Figaro, please refer to the tutorial document that comes with the distribution. [5]

Notice from Table 1 that rocket bodies have an "Uncontrolled" attitude profile by definition. It is often more direct to use a Constant representation for this type of characteristic because it will be assumed by the Figaro reasoning engine to be constant for all time. However, if you want to allow for the fact that your observations could indicate a controlled attitude at some point, then you would need to use the Select representation and update the underlying

probability distribution. For this example, we are assuming that Payloads have constant "low" area-to-mass ratios while rocket bodies and debris fragments both have constant "uncontrolled" attitude and spin states. Since we are assuming that a payload has an equal probability of assuming a particular attitude and spin state, we used the Uniform distribution for each of these characteristics. Because the area-to-mass ratio for debris objects can vary, we used a Select representation to demonstrate a varied probability for each broad category.

To demonstrate the more elegant features of Figaro using class inheritance and rich data structures, we will expand the example taxonomy slightly to include a few object mission types. This class structure can be seen graphically in Figure 10, and, along with the characteristic features described in Table 1, can be represented using the Scala language as shown in Figure 11 and continued in Figure 12. In Figure 13, we provide the commands to execute the Figaro reasoning engine. For detailed information about these commands, we refer the reader again to the Figaro documentation. [5]

Table 1: Example RSO taxonomy characteristics and associated probabilities.

		Probability of Occurrence		
Characteristic	Value	Payload	Rocket Body	Debris Fragment
Area-to-Mass Ratio	Low/Intact	1	1	0
	Medium/Fragment	0	0	0.75
	High/Fragment	0	0	0.25
Attitude Control	Uncontrolled	0.5	1	1
	Controlled	0.5	0	0
Spin State	Uncontrolled	0.25	1	1
	Spin Stabilized	0.25	0	0
	Nadir Pointing	0.25	0	0
	Sun Pointing	0.25	0	0

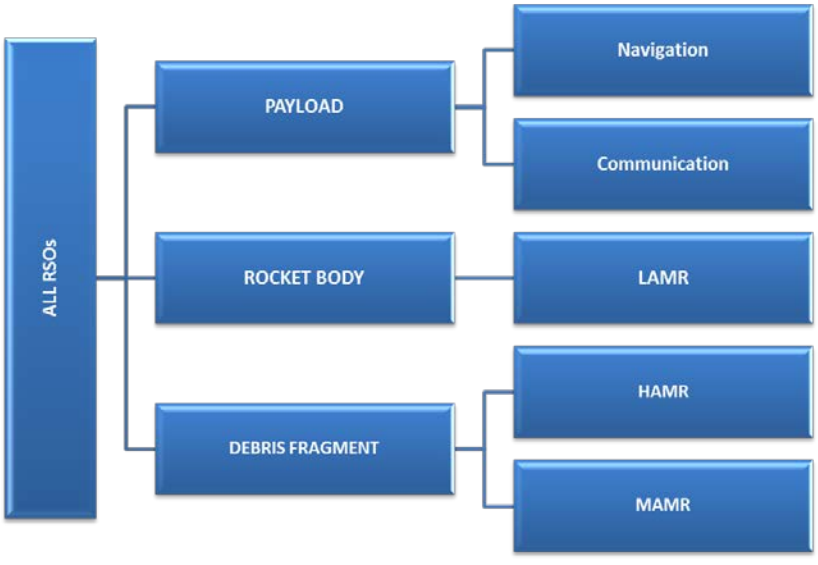

Figure 10: Example taxonomy showing class inheritance.

```
import com.cra.figaro.language._
import com.cra.figaro.algorithm.factored.VariableElimination
import com.cra.figaro.library.atomic.discrete.Uniform

object RSOHierarchy {

  val universe = Universe.createNew()

  abstract class RSO extends ElementCollection {
    val speed: Element[Double]
    val amr: Element[Symbol]
    val attitude: Element[Symbol]
    val spin: Element[Symbol]
    val shape: Element[Symbol]
  }

  class Payload extends RSO {
    val amr: Element[Symbol] = Select(0.9 -> 'low, 0.1 -> 'medium, 0.0 -> 'high)("amr", this)
    val attitude: Element[Symbol] = Select(0.5 -> 'uncontrolled, 0.5 -> 'controlled)("attitude", this)
    val spin: Element[Symbol] = Select(0.25 -> 'uncontrolled, 0.25 -> 'spinstable,
                                       0.25 -> 'nadir, 0.25 -> 'sun)("spin", this)
    val shape: Element[Symbol] = Uniform('regular, 'irregular)("shape", this)
    val speed: Element[Double] = Uniform(7.0, 5.0, 3.0)("speed", this)
  }

  class Navigation extends Payload {
    override val speed: Element[Double] = Constant(5.0)("speed", this)
    override val amr: Element[Symbol] = Constant('medium)("amr", this)
  }

  class Communication extends Payload {
    override val speed: Element[Double] = Constant(3.0)("speed", this)
    override val amr: Element[Symbol] = Constant('tiny)("amr", this)
  }

  class RocketBody extends RSO {
    val amr: Element[Symbol] = Constant('low)("amr", this)
    val attitude: Element[Symbol] = Constant('uncontrolled)("attitude", this)
    val spin: Element[Symbol] = Constant('uncontrolled)("spin", this)
    val shape: Element[Symbol] = Constant('rocketbody)("shape", this)
    val speed: Element[Double] = Uniform(7.0, 5.0, 3.0)("speed", this)
  }

  class DebrisFragment extends RSO {
    val amr: Element[Symbol] = Select(0.0 -> 'low, 0.75 -> 'medium, 0.25 -> 'high)("amr", this)
    val attitude = Constant('uncontrolled)("attitude", this)
    val spin = Constant('uncontrolled)("spin", this)
    val shape = Constant('debris)("shape", this)
    val speed: Element[Double] = Uniform(7.0, 5.0, 3.0)("speed", this)
  }

  class LAMR extends RocketBody {
    override val amr: Element[Symbol] = Constant('low)("amr", this)
    override val speed: Element[Double] = Uniform(7.0, 5.0, 3.0)("speed", this)
  }

  class MAMR extends DebrisFragment {
    override val amr: Element[Symbol] = Constant('medium)("amr", this)
    override val speed: Element[Double] = Uniform(7.0, 5.0, 3.0)("speed", this)
  }

  class HAMR extends DebrisFragment {
    override val amr: Element[Symbol] = Constant('high)("amr", this)
    override val speed: Element[Double] = Constant(3.0)("speed", this)
  }

  <continued next page…>
```

```
object RSO {
  def generate(name: String): Element[RSO] = Dist(0.08 -> RocketBody.generate,
      0.34 -> Payload.generate, 0.58 -> DebrisFragment.generate)(name,universe)
}

object Payload {
  def generate: Element[RSO] = Dist(0.56 -> Communication.generate,
                                    0.09 -> Navigation.generate,
                                    0.35 -> Constant[RSO](new Payload))
}

object Navigation {
  def generate: Element[RSO] = Constant(new Navigation)
}

object Communication {
  def generate: Element[RSO] = Constant(new Communication)
}

object RocketBody {
  def generate: Element[RSO] = Constant(new RocketBody)
}

object DebrisFragment {
  def generate: Element[RSO] = Dist(0.7 -> MAMR.generate, 0.25 -> HAMR.generate)
}

object MAMR {
  def generate: Element[RSO] = Constant(new MAMR)
}

object HAMR {
  def generate: Element[RSO] = Constant(new HAMR)
}

// A helper function to retrieve class names
def shortClassName[T](elem: Element[T]): Element[String] =
  Apply(elem, (t: T) => {
    val fullName = t.getClass().getName()
    val dollar = fullName.indexOf('$')
    fullName.substring(dollar + 1, fullName.length)
  })

}
```

Figure 12: Scala code for the RSO taxonomy companion objects.

The code has been broken up into chunks for discussion purposes only. In order to compile the code, one will need to combine the code snippets from Figure 11 through Figure 13 into a single file. The first code snippet, presented in Figure 11, contains characteristic data for each class of object. Each characteristic has been assigned a probabilistic distribution per the discussion above. Notice that the class hierarchy is maintained by use of the Scala *extends* keyword. In Figure 12, Scala companion objects are instantiated for each class member in the hierarchy. These companion objects are used by the Figaro engine to construct a distribution of child classes in the hierarchy. For example, the code snippet for the RSO companion object

```
object RSO {
     def generate(name: String): Element[RSO] = Dist(0.08 -> RocketBody.generate,
        0.34 -> Payload.generate, 0.58 -> DebrisFragment.generate)(name,universe)
   }
```

tells the Figaro engine to generate a RocketBody with 8% probability, a Payload with 34% probability, and a DebrisFragment with 58% probability. In reality, the probabilistic distributions provided here would be learned and updated as new evidence is presented. For clarity and length, we have chosen to present these values as constants in

this example. On a very basic level, this tells Figaro that if you have no other evidence, this is the distribution of object types. Likewise, each successive companion object in the overall hierarchy will have its own separate distribution of child objects that will serve to further refine the results from Figaro's reasoning engine.

```
  def main(args: Array[String]) {

    val myRSO = RSO.generate("rso1")
    val name = shortClassName(myRSO)
    val i1 = Apply(myRSO, (r: RSO) => r.isInstanceOf[Communication])
    universe.assertEvidence(List(NamedEvidence("rso1.amr", Observation('tiny))))
    val alg = VariableElimination(i1, name)
    alg.start()
    println(alg.probability(i1, true))
    println(alg.distribution(name).toList)

    universe.assertEvidence(List(NamedEvidence("rso1.amr",
         Observation('tiny)),NamedEvidence("rso1.speed", Observation(3.0))))
    val i2 = Apply(myRSO, (r: RSO) => r.isInstanceOf[Navigation])
    val alg2 = VariableElimination(i2, name)
    alg2.start()
    println(alg2.probability(i2, true))
    println(alg2.distribution(name).toList.sorted)
  }
} // End of Main
} // End of RSOHierarchy Object
```

Figure 13: Scala code demonstrating how to condition the model with evidence as well as how to reason with the model.

The most vital aspect of the entire taxonomy example is presented in Figure 13. Here the elegance of the Figaro reasoning engine begins to shine through. Once the RSO taxonomy has been established and the *a priori* characteristic probabilities have been defined, the ability to reason with the model is relatively easy. One must simply issue a command to "*assertEvidence*" which could be a list of symbolic, numeric, or string data.

```
universe.assertEvidence(List(NamedEvidence("rso1.amr", Observation('tiny))))
```

One could just as easily return evidence from a multiple model adaptive estimator (MMAE) through a function call to other aspects of your code base. For example, an interface function to your MMAE could return the most likely spin state as evidence.

```
universe.assertEvidence(List(NamedEvidence("rso1.spin",
        Observation((spin: Symbol) => kalmanFilterBankEstimate(spin)))))
```

Once the evidence has been asserted, one could use Figaro's variable elimination algorithm, among others, to generate reasoned results. To use variable elimination, one must specify a set of query elements whose conditional probability you want to compute given the evidence. This is accomplished by first specifying a "*universe*" which contains all the taxonomic classes and objects with their associated characteristics. The command in the example tells Figaro to solve the variable elimination problem by elimination of all variables *except* for the ones listed (*i1*, *name*). In this case, the variable *i1* is a way to assert that you think the object "rso1" is actually a communication satellite. The variable *name* is simply a list of class names in our taxonomy.

```
val i1 = Apply(myRSO, (r: RSO) => r.isInstanceOf[Communication])
val alg = VariableElimination(i1, name)
alg.start()
```

The variable elimination process is not begun and results cannot be obtained until the *start* command has been issued. The remaining lines of code tell Figaro to output desired information to the command line in this case. *alg.probability(i1, **true**)* will output the probability that the object *i1*, which was assigned to be an instance of a

Communication satellite, is actually a Communication satellite given the evidence. *alg.distribution(name)* provides the distribution of each class of object in the class hierarchy given the evidence.

10. EXAMPLE RESULTS

The results from compiling and executing the above code using the Scala development environment are presented in Figure 14. In this example, we have asked Figaro the same question in two different ways and received back useful probabilistic data. First, we asked if RSO1 was a Communication satellite according to our probabilistic model given that its area-to-mass ratio is *tiny*. Since Communication satellites are the only object with size designated as *tiny*, the answer was a 100% chance of that being a correct identification given the evidence. Second, we asked for the distribution of all object types given the evidence. Here one can see that the Communication class is included in the list along with all other classes in the taxonomy. We then posed the question again but this time asserted that the object was a Navigation satellite. Additionally, we asserted that the speed of the object had been observed. This time Figaro responds that the probability of the object being a Navigation satellite is 0%, and the distribution of object types remains the same because the additional speed information simply confirms that the object is actually a Communication satellite.

```
Probability RSO1 is a Communication Satellite: 1.0

Ranked Hypotheses for Object Type Given the Evidence:

List((1.0,Communication),  (0.0,HAMR),  (0.0,DebrisFragment),  (0.0,MAMR),
(0.0,Payload), (0.0,RocketBody), (0.0,Navigation))

Probability RSO1 is a Navigation Satellite: 0.0
Ranked Hypotheses for Object Type Given the Evidence:

List((1.0,Communication),  (0.0,HAMR),  (0.0,DebrisFragment),  (0.0,MAMR),
(0.0,Payload), (0.0,RocketBody), (0.0,Navigation))
```

Figure 14: Results from executing the Figaro reasoning example.

Now, let us change up the example ever so slightly. We now change the definition of Communication satellite area-to-mass ratio from *tiny* to *high*, which would put a Communication in the same size category as a High Area-to-Mass Ratio (HAMR) Debris Fragment.

```
class Communication extends Payload {
    override val speed: Element[Double] = Constant(3.0)("speed", this)
    override val amr: Element[Symbol] = Constant('high)("amr", this)
   }
```

We also change the asserted evidence in Figure 13 from *tiny* to high as shown in Figure 15. The results of this second example are presented in Figure 16.

```
   def main(args: Array[String]) {

        val myRSO = RSO.generate("rso1")
        val name = shortClassName(myRSO)
        val i1 = Apply(myRSO, (r: RSO) => r.isInstanceOf[Communication])
        universe.assertEvidence(List(NamedEvidence("rso1.amr", Observation('high))))
        val alg = VariableElimination(i1, name)
        alg.start()
        println(alg.probability(i1, true))
        println(alg.distribution(name).toList)

        universe.assertEvidence(List(NamedEvidence("rso1.amr",
         Observation('high)),NamedEvidence("rso1.speed", Observation(3.0))))
        val i2 = Apply(myRSO, (r: RSO) => r.isInstanceOf[Navigation])
        val alg2 = VariableElimination(i2, name)
        alg2.start()
        println(alg2.probability(i2, true))
        println(alg2.distribution(name).toList.sorted)
   }
} // End of Main
} // End of RSOHierarchy Object
```

Figure 15: A second example Scala code demonstrating how to condition the model with new evidence.

```
Probability RSO1 is a Communication Satellite: 0.28099173553719015

Ranked Hypotheses for Object Type Given the Evidence:

List((0.4793388429752066,HAMR), (0.28099173553719015,Communication),
(0.2396694214876033,DebrisFragment), (0.0,MAMR), (0.0,Payload),
(0.0,RocketBody), (0.0,Navigation))

Probability RSO1 is a Navigation Satellite: 0.0
Ranked Hypotheses for Object Type Given the Evidence:

List((0.5704918032786885,HAMR), (0.3344262295081968,Communication),
(0.09508196721311477,DebrisFragment), (0.0,MAMR), (0.0,Payload),
(0.0,RocketBody), (0.0,Navigation))
```

Figure 16: Results from executing the second Figaro reasoning example.

The implications of changing the probabilistic model can be seen clearly in this second example. Now that the area-to-mass ratio of the Communication satellite has been changed to match that of other objects, the probabilistic answer to whether this object belongs to the class of Communcation objects is less certain. When one adds in the additional speed observations, the uncertainty drops from 28% to 0%. Looking at the list of possible object types, the available evidence of *amr* and *speed* suggest that the object is most likely a HAMR Debris Fragment.

11. CONCLUSIONS

In this work, we have provided the basis for a resident space object taxonomy. One of the key aspects of this work is that taxonomic hierarchies present a way to determine a unique signature for an object in the form of the minimum amount of information needed to correctly recognize and identify an object. We have formulated the taxonomy to incorporate sensor data, dynamic model choices, as well as the physical characteristics of the RSOs. This taxonomy allows one to both model the appearance of objects as well as reason with observations in a Bayesian construct. We have presented the Figaro probabilistic programming language for this purpose and demonstrated how to implement a simple taxonomy. Furthermore we showed a simple example of the Figaro reasoning engine to ask probabilistic questions regarding the state and nature of an observed RSO. Not only can we ask specific questions and receive

answers with associated confidence but also we can generate lists of hypotheses that can be ranked according to their confidence level.

12. ACKNOWLEDGEMENTS

The authors would like to acknowledge the sponsorship of the Air Force Research Laboratory for supporting this researching.

13. REFERENCES

1. *Development of an Initial Taxonomy and Classification Scheme for Artificial Space Objects.* **Früh, C., et al., et al.** Darmstadt, Germany : s.n., 2013. Sixth European Conference on Space Debris.
2. **Milch, B., et al., et al.** BLOG: Probabilistic Models With Unknown Objects. [book auth.] L. Getoor and B. Taskar. *Statistical Relational Learning.* s.l. : MIT Press, 2007.
3. *Identity Uncertainty and Citation Matching.* **Pasula, H., et al., et al.** 2003. Proceedings of Neural Information Processing Systems.
4. *A Probabilistic Rational Programming Language.* **Pfeffer, Avi.** 2012. Proceedings of Workshop on Statistical Relational Artificial Intelligence (StarAI).
5. Figaro Software and Tutorial. *Charles River Analytics.* [Online] [Cited: June 26, 2013.] Available for download directly from Charles River Analytics: https://www.cra.com/commercial-solutions/probabilistic-modeling-services.asp.
6. Naming Astronomical Objects. *International Astronomical Union.* [Online] [Cited: June 26, 2013.] http://www.iau.org/public/naming/.
7. **Gray, Richard O. and Corbally, Christopher J.** *Stellar Spectral Classification.* s.l. : Princeton University Press, 2009.
8. *Long Term Analysis of GEO Photometric Signatures.* **Payne, Tamara, Gregory, Stephen and Houtkooper, Nina.** Wailea, HI : s.n., 2003. 2003 AMOS Technical Conference Proceedings.
9. *Three-Dimensional Analysis of GEO Photometric Signatures.* **Payne, Tamara and Gregory, Stephen.** Wailea, HI : s.n., 2004. 2004 AMOS Technical Conference.
10. *Electro-optical Signatures Comparisons of Geosynchronous Satellites.* **Payne, T.E., Gregory, S.A. and Luu, Kim.** Big Sky, MT : IEEE, 2006. Aerospace Conference.
11. *SSA Analysis of GEOS Photometric Signature Classifications and Solar Panel Offsets.* **Payne, Tamara, Gregory, Stephen and Luu, Kim.** Wailea, Maui, HI : s.n., 2006. 2006 AMOS Tech Conference.
12. Satellite Monitoring, Change Detection, and Characterization Using Non-resolved Electro-Optical Data from a Small Aperture Telescope. **Payne, Tamara, et al., et al.** Wailea, Maui, HI : s.n., 2007. 2007 AMOS Tech Conference.
13. UCS Satellite Database (12-1-12). *Union of Concerned Scientists.* [Online] [Cited: June 6, 2013.] http://www.ucsusa.org/satellite_database.
14. *Church: a Language for Generative Models.* **Goodman, N. D., et al., et al.** 2008. Proceedings of Uncertainty in Artificial Intelligence.
15. *Effective Bayesian Inference for Stochastic Programs.* **Koller, D., McAllester, D. and Pfeffer, A.** 1997. Proceedings of National Conference on Artificial Intelligence (AAAI).
16. *IBAL: A Probabilistic Rational Programming Language.* **Pfeffer, A.** 2001. Proceedings of International Joint Conference on Artificial Intelligence.