Framework for Ontology-Driven Decision Making

Kenneth Baclawski[*] Eric S. Chan[†] Dieter Gawlick[‡]
Adel Ghoneimy[‡] Kenny Gross[§] Zhen Hua Liu[‡]
Xing Zhang[¶]

November 3, 2017

Abstract

Decision making is an important part of human activities, and is a very active area of research. In this article, a formal framework for decision making is developed. At the heart of the framework is an ontology for the decision making process that classifies both the kinds of data that are available for the decision making process and the modes of reasoning that are used to develop situation awareness and to make decisions. This ontology-driven framework resolves two outstanding problems in decision making systems: providing a theoretical foundation and declarative language support. Concerns are addressed such as providing for multiple notions of time, recording the activities used in decision making, and maintaining the provenance of both data and activities. A process model is developed that allows for flexible execution of the activities, including either synchronous or asynchronous processing and for exploring alternative or concurrent branches at any stage of the process. Detailed implementation techniques are presented. An issue tracking system is used as a running example to illustrate the concepts being developed. Use cases for the framework are given in a variety of domains including customer support, healthcare, cloud services, and the Internet of Things.

Keywords: decision making, situation awareness, information fusion, customer service, provenance, issue tracking systems

1 Introduction

The purpose of the Ontology Summit 2014 "Big Data and Semantic Web Meet Applied Ontology" was to build bridges between the Semantic Web, Linked Data, Big

[*]Northeastern University, Boston, MA USA
[†]Workday, Inc., Pleasanton, CA USA
[‡]Oracle Corporation, Redwood City, CA USA
[§]Oracle Corporation, San Diego, CA USA
[¶]College of Engineering, Northeastern University, Boston, MA USA

Data, and Applied Ontology communities. There was a special concern that ontologies appear to have had little impact in Big Data applications, and one of the tracks of Ontology Summit 2014 dealt specifically with Big Data. Big Data applications are often characterized by the three V's: Volume, Velocity and Variety. The last of these, Variety, was recognized as having the potential for significantly benefiting from a greater use of ontologies, so it was the focus of the Big Data Track of Ontology Summit 2014 (Baclawski and Thessen, 2014). The process employed in Big Data makes use of statistical techniques for discovery and hypothesis testing; in other words, it is a scientific discovery process. One of the topics discussed in this track was the problem of computer-based decision making. The purpose of this article is to expand on this topic from the Big Data track at Ontology Summit 2014, by developing a bridge between Big Data processes and the Semantic Web, Linked Data and Applied Ontology.

Decision making necessarily occurs within a context (or, more precisely, a situation) and requires that one have a purpose or goal that the decision making process is intended to help achieve, although the goal is often implicit. The goal serves both to direct the decision and to infer the facts that are relevant to the situation. Directing the decision and inferring relevance may be regarded as proceeding at different levels. The former occurs at the level of facts (A-box) while the latter occurs at a higher level (T-box). Constructing the situation within which a decision will be made is called "situation awareness." It is "a motivated, continuous effort to understand connections (which can be among people, places, and events) in order to anticipate their trajectories and act effectively (Klein, Moon, & Hoffman, 2006a)."

Because decision making occurs on multiple levels, it is an active bidirectional process of fitting data into a frame and fitting a frame around the data. A frame for representing data is its schema. When enriched with formal semantics, a schema is an ontology. Thus the process of situation awareness not only processes raw observations to produce data represented within an ontology, but also creates the ontology within which the data is to be represented (Klein, Moon, & Hoffman, 2006b).

Although situation awareness and decision making have been a topic of research for many years, the emphasis is on human interaction rather than the formal process of decision making. The most prominent example of the former is John Boyd's Observe, Orient, Decide, Act (OODA) loop (Boyd, 1976), but there are many others. The Joint Directors of Laboratories' Data Fusion Information Group (JDL/DFIG) model (Steinberg, Bowman, & White, 1999; Steinberg & Bowman, 2001) is a standard for data fusion that includes situation assessment as one of a series of levels. The JDL/DFIG model includes process models but is only descriptive. The Knowledge Intensive Data System (KIDS) (Gawlick, Chan, Ghoneimy, & Liu, 2015) is a formal framework based on the OODA loop that provides for:

1. Categorization of data and reasoning processes.
2. State management of data, reasoning and process, and
3. Process model for situation awareness and decision making.

In this article, we briefly survey the frameworks for situation awareness and decision making in Section 2. We then introduce a running example of an issue tracking

system in Section 3 that will be used to help explain the KIDS framework and the corresponding ontology for decision making. The KIDS framework is described in Section 4, and the KIDS ontology is presented in Section 5. The KIDS ontology solves two of the outstanding challenges in Gawlick et al., Section 6; namely, Theoretical Foundations and Declarative Language Support. In addition, the KIDS ontology introduces a more general execution model than Gawlick et al.; and integrates the KIDS model with the PROV-O provenance ontology (PROV-O, 2013). The KIDS ontology is designed to allow for both data and process diversity. These features of our ontology-driven decision making framework are discussed in Section 6 where we describe some of the challenges and give detailed implementation techniques for our framework. To show the efficacy of the KIDS framework and ontology, in Section 7 we give a number of significant use cases where the framework can be employed. We end with conclusions in Section 8.

2 Background

There are far too many decision making tools for even a superficial survey within the scope of this article. Ontologies for decision making, on the other hand, are much less common. To the extent that there are such ontologies, they focus on decision criteria and alternatives rather than on the process of decision making. In other words, these ontologies record the *rationale* for a decision. Rationales are an important part of modern project development, and represent a mechanism for recording how decisions were made so that they can be revisited when circumstances change significantly (Bruegge & Dutoit, 2009). The earliest example of a rationale ontology is in Duggar and Baclawski (2007). There is a tool based on this ontology for managing rationales and integrating them in a software development project, specifically with the Eclipse IDE (Duggar, 2008). While the Rationale Ontology is useful for recording the requirements and alternatives that were considered when making a decision, it is primarily concerned with collaborative decision making by people and does not deal with the decision making process. There are other rationale ontologies such as the Decision Making (DM) ontology in Kornyshova and Deneckere (2010) which is similar to the Rationale Ontology in Duggar and Baclawski. As we explain below, situation awareness is an essential part of any decision making process. While the DM ontology article mentions situations, it does not deal with situation awareness, does not cite work on situation theory or the earlier work on the Rationale Ontology, does not integrate the DM ontology with any closely related existing ontologies, and does not mention any tools based on their ontology.

The fundamental concept for decision making is the notion of a situation. A situation represents a limited part of reality that can be perceived and reasoned about. A situation will generally be about some objects within a spatio-temporal extent. Situation theory was developed by Barwise (1981); Barwise and Perry (1983); Barwise (1989), and subsequently extended by Devlin (1991). The Situation Theory Ontology (STO) was developed in Baclawski, Malczewski, Kokar, Letkowski, and Matheus (2002); Baclawski, Kokar, Matheus, Letkowski, and Malczewski (2003); Matheus, Kokar, and Baclawski (2003) and is available at Baclawski, Malczewski, Kokar, Letkowski, and

Matheus (2006). The STO is specified using the Web Ontology Language (OWL). The information of that part of reality being represented by a situation is specified using *infons*, which are logical sentences about objects relevant to the situation. An infon can assert that a relationship either holds or does not hold among some objects. This is called an elementary infon, and it corresponds to an atomic formula in predicate logic. Other kinds of infon are discussed below. An important feature of STO is that a situation is a first-class object that can, in principle, be part of another situation. However, STO does not address the processes involved in dealing with situations. STO has other difficulties that prevent it from being used in the KIDS ontology. STO has meta-classes that have classes as their instances. This is inconsistent with Description Logic (DL).[1] Consequently, OWL DL editors, such as Protégé (Protege, 2015), and OWL DL reasoners, such as FaCT++ (Tsarkov & Horrocks, 2006) cannot properly process STO. Another difficulty with STO is that it does not support the general notion of a situation theory infon, being limited to just elementary infons.

The main class of the STO is the Situation class. This class has three subclasses: utterance situation, resource situation and focal situation. An utterance situation consists of queries or other reasoning processes. While these are traditionally considered to be statements spoken by a person (and hence the use of the word "utterance" for this kind of situation), they can be "uttered" by any agent, whether the agent is a person or a software system. A resource situation consists of data that serves as the general background for reasoning with other situations. A focal situation or *described situation* is that part of the world (as formalized by the resource situation) that is relevant to a given utterance situation. As Devlin explains, "Also known as the described situation, the focal situation is that part of the world the utterance is about. Features of the utterance situation serve to identify the focal situation Devlin, page 12."

In situation theory, information about a situation is expressed in terms of *infons.* Infons may be recursively combined to form *compound infons* by using conjunction, disjunction and situation-bounded quantification. Devlin states that "infons are not things that in themselves are true or false. Rather a particular item of information may be true or false about a situation Devlin, page 3." We also allow infons to be probabilistically specified using a joint probability distribution. Such a distribution can be efficiently specified using Bayesian networks (Pearl, 2000). The relationship between a situation s and an infon σ is called the *supports* relationship and it is written

$$s \models \sigma.$$

The symbol used for the supports relationship is the same as the symbol used for logical entailment, but the supports relationship can be computed in a variety of ways. As we will see in Sections 4 and 5, the decision making process will generally use one situation to produce another one, which is not necessarily a monotonic process, although any reasoning within a situation may remain monotonic.

The infons in a situation include time information as well as the "polarity" which indicates whether the fact is true or false in the situation. In some cases, several times

[1] OWL has a technique for dealing with many modeling use cases that seem to require meta-classes. This technique is called "punning." However, punning is effective only when there is no expectation for any logical consequences of meta relationships (Punning, 2007). Since STO does expect the meta relationships to have logical consequences, punning cannot be used.

are specified, such as a database transaction timestamp and the time interval when the infon is deemed to be valid. For example, infons supported by a situation with a limited temporal extent will generally have a valid time interval. For example,

$$s \models \ll chases, Rex, Fluffy, l, v, t, true \gg$$

says that in the situation s, Rex chases Fluffy in the location l and valid time interval v, and database timestamp t, with polarity equal to true. One can, by using a suitable query, extract the state of the situation (and hence a part of the world) at a specific instant of time. Consequently, situations are a mechanism for managing the state of a part of the world, both observing the state and modifying it.

A prototype situation awareness assistant (SAWA) (Matheus et al., 2005b) was developed that provided some valuable lessons about the problems one encounters when one automates the process of situation assessment (Matheus et al., 2005a). SAWA is based on STO and uses OWL to represent facts. The first lesson is that rules are more complicated in OWL, which can only represent binary relations, compared with representation languages that support ternary and higher-order relations, because representing higher-order relations in OWL requires them to be reified. However, representing infons and situations generally requires reification anyway in order to represent the STO "supports" relation, so this problem is not as big as it might be for other applications. Some other lessons mentioned in Matheus et al. were concerned with temporal reasoning and uncertainty propagation. Another limitation of SAWA was that it did not maintain provenance information. We propose mechanisms for dealing with these lessons.

While STO deals with situational inference, it does not elaborate on the process whereby inference occurs. In addition, STO has other difficulties that are noted above. So the KIDS ontology does not import STO, although KIDS does include most of the notions in STO. However, the KIDS ontology can be regarded as an extension of situation theory that formalizes the decision making process. Our framework uses the OODA loop Boyd as the basis for the decision making process. In this loop, as illustrated in Figure 1, experts *Observe* the facts by capturing and filtering relevant data about the entities and environment, *Orient* the human participants to the information condensed from the facts by applying knowledge and personalization (which are analogous to Boyd's notions of cultural traditions and genetic heritage), *Decide* on the directives based on the hypotheses that best explains the observations, and *Act* on the directives to interact with the entities and environment and to test the hypothesis.

The OODA loop is popular in many industries because rapid iteration of the OODA loop allows one to continuously interact with the environment to assess and adapt to uncertainty, incomplete knowledge and changes in the environment Boyd. The original purpose of the OODA loop was for improving decision making by fighter pilots. The OODA loop has since been used extensively in many domains such as warfare in general (Osinga, 2006), business decision making (Richards, 2004), litigation (Dreier, 2012), and healthcare delivery (Toner, 2010; Fioratou, Flin, Glavin, & Patey, 2010; Healthcare, 2016). The OODA loop is similar to other decision cycles such as scientific discovery (Darian, 2003). However, nearly all of the literature on the OODA loop is concerned with the decision making process of people rather than computers.

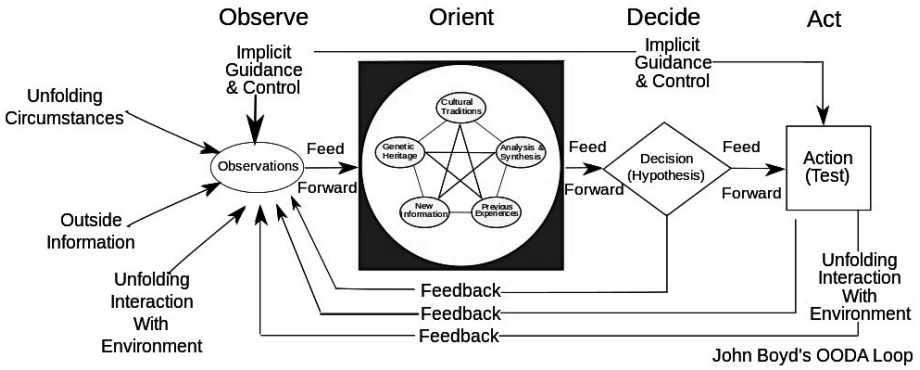

Figure 1: Diagram of the decision cycle known as the Boyd cycle, or the OODA loop (Moran, 2008)

As Georgakopoulos (2008) pointed out, "... there are few software systems that are capable of supporting OODA." Georgakopoulos went on to propose an architecture for supporting OODA. While the OODA loop can be regarded as an architecture for decision making, it has no formal treatment of either data or how reasoning takes place. In particular, there is no explicit model of the data, as it is presumed that all of this is in the mind of the human making the decision. For the OODA loop to be applicable to computer-based decision making processes, it must be further refined and expanded to formalize the data and reasoning processes as well as how these are used to make decisions.

As noted above, provenance is an important requirement of any decision making process. In order to maintain provenance information, we use the PROV-O ontology which is a W3C Recommendation for representing provenance information PROV-O. To represent the workflow aspects of KIDS we use the OASIS Integrated Collaboration Object Model for Interoperable Collaboration Services (ICOM) which defines a framework for integrating a broad range of domain models for collaboration activities in an integrated and interoperable collaboration environment (ICOM, 2013). The advantage of ICOM over other workflow models is the support for flexible collaboration using tools such as email, teleconferencing, chat, wikis, and social networking. Other workflow models are generally pre-defined and linear.

3 Issue Tracker Example

To illustrate our framework, we will use a running example. This example is an issue tracking system. Such systems are used for dealing with bugs in the software development lifecycle, but they are applicable to many other development processes. An issue tracking system manages issues raised by individuals who have a stake in some product. For example, the customers of a company or the users of a software artifact. Issues have a life-cycle in which the issues are reported, tracked and eventually resolved. A particular instance of an issue raised by a participant is often called a "ticket." A ticket

includes all the information related to the progress toward resolving the issue. A ticket may be regarded as an evolving situation. The handling of an issue may require a series of cycles if it cannot be immediately resolved. In such a process there may therefore be a series of decisions and actions which help to resolve the issue.

An important aspect of any issue tracking system, as its name suggests, is the requirement that an issue be *tracked*. In other words, one should manage not only the data associated with the steps in issue resolution but, ideally, also the provenance of the data. For example, it is important to track not only the decision that was taken but also the process that led to the decision.

The process of issue tracking and resolution can be defined as having the following steps:

1. The issue is raised by a reporter. The reporter describes the issue and provides related data.
2. The issue is classified. Typically there is a fixed set of categories, but there are usually mechanisms for introducing customized categories. Issues are also often assigned priorities as well as other attributes.
3. The possible causes are identified. This may require its own process.
4. Decisions are made in an effort to fully or partially resolve the issue. In elaborate cases, this will involve a detailed plan.
5. Actions are taken to carry out the decisions.

In the rest of this article, we will relate the steps above to the KIDS model with its decision making process.

4 The KIDS Framework

At the highest level, the KIDS framework is a decision making and enactment process for situations that distinguishes different kinds of data as well as different kinds of reasoning in the decision making process (Gawlick et al., 2015; Liu, Behrend, Chan, Gawlick, & Ghoneimy, 2012; Chan, Behrend, Gawlick, Ghoneimy, & Liu, 2012; Chan, Gawlick, Ghoneimy, & Liu, 2014). In this section we describe these kinds of data and reasoning as well as how they fit together in the decision making process.

4.1 The KIDS Data Hierarchy

In the KIDS framework, a situation is represented by a collection of instances of the Data class Chan et al.. These instances are the infons supported by a situation. The Data class has four disjoint subclasses, depending on how the data is employed in the process of decision making and enactment. This is in contrast to the generic situation theory ontology Baclawski et al., which does not subclassify infons or address the various steps in the decision making process. The four subclasses of Data are Fact, Perception, Hypothesis and Directive. We now discuss what these subclasses represent.

In the KIDS framework, the Fact subclass represents observations of reality. Current data collection tools produce data at a rate that makes it impossible for human cognition to deal with directly. To start a decision making process, it is necessary to perform relevance reasoning to select the infons in the first situation from all of the observed data. Modern sensor technology produces very large amounts of data, only a small part of which will be relevant to a particular purpose. In our running example of an issue tracking system, these observations are, at least initially, provided by the issue reporter. Additional observations are added as the situation evolves. For software bug tracking, the observations may be system traces or other data obtained by monitors.

The Perception subclass represents compact interpretations of the facts, obtained by a data reduction process. Most commonly, perceptions are classifications of the facts. Unlike raw facts, perceptions are sufficiently compact to be directly dealt with by humans. Perceptions are the most important characteristics of an evolving situation. For an issue tracking system, perceptions are the classifications of the observations. Typically, the reporter is not only responsible for collecting the raw facts for the issue but is also responsible for the initial classification of the facts. This is necessary, in practice, because the reporter is a person who is raising the issue to other persons. So communicating with raw facts alone would not be feasible. For software bug tracking, a perception is a statement of the problem that was encountered.

The Hypothesis subclass represents possible causes that explain the facts and perceptions. In practice, there will be many possible causes, and it can be difficult to distinguish them. In some situations, there may even be controversy associated with suggesting different causes. Unlike perceptions, hypotheses are not essential to a decision making process. One could directly make decisions without determining the causes for the perceptions. For an issue tracking system, hypotheses are the possible underlying reasons for the perceived bugs.

The Directive subclass represents the action plans that are proposed to address the perceptions and hypotheses. A directive could fully address the situation or it might only partially address it. A directive might, for example, be a plan for gathering additional observations rather than directly resolving the situation. For an issue tracking system, a directive is either a plan for correcting the issue, a plan for collecting more data about the situation or both or neither. Note that a directive can explicitly state that the issue will not be addressed and will be closed. This can be due to a variety of reasons such as limited resources and pressure to release new product versions with new features. In the case of bug tracking, one might try to patch the software while at the same time running tests to clarify whether the proposed causes have been correctly identified.

4.2 The KIDS Reasoning Hierarchy and CARE Loop

Having classified the data according to how it is used, we now classify how reasoning is performed. We refer to the different types of reasoning that occur in the process of decision making and enactment as the *modes* of reasoning. For each mode of reasoning, we distinguish the process being applied from the action in which a process is applied, which includes parameters and annotations such as the time when the action was applied and the execution environment. The execution environment includes,

for example, the operating system and version information about the programs employed. The purpose of recording the execution environment is to provide background information sufficient for reproducing the result of the activity invocation. One can think of a reasoning activity as being an example of the Command Design Pattern of object-oriented software development (Gamma, Helm, Johnson, & Vlissides, 1995). Collectively, the modes of reasoning and enactment employed in KIDS are instances of the Reasoning class of KIDS, while the reified reasoning processes are instances of the Activity class which is borrowed from the PROV-O ontology. The general notion of a "situation" includes both data and reified reasoning processes.

The Reasoning class has six disjoint subclasses: Relevance, Classification, Assessment, Resolution, Assessment/Resolution and Enactment. The Activity class has six disjoint subclasses corresponding to the subclasses of Reasoning: Filter, Classify, Assess, Resolve, Assess/Resolve and Enact. The six kinds of activity and four kinds of data form a decision making and enactment process loop as shown in Figure 2. This loop is called the CARE Loop after the names of the four main classes of activity being performed. The CARE Loop can be configured to run with a fixed time period for each loop or to run in response to events, depending on the requirements of the application domain.

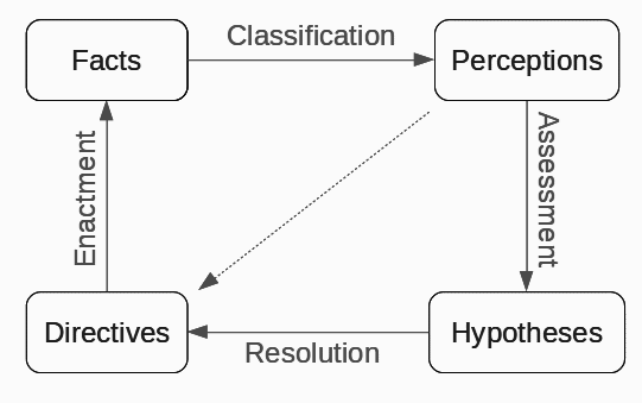

Figure 2: Diagram of the decision cycle known as the CARE Loop of the KIDS Framework

An instance of Classification is a process that transforms raw facts into perceptions. While the most common form of reasoning is deductive, it is also possible to use inductive reasoning. Examples of such classification processes include Support Vector Machines, Naïve Bayesian Networks, Neural Networks, Clustering, Association Rules, decision trees, Multivariate State Estimation Techniques (MSET), etc. For the issue tracking example, relatively simple classification processes are commonly used, including ad hoc queries and filters and even human cognition, although some systems employ more sophisticated classification techniques.

The Assessment subclass represents the processes that transform perceptions into hypotheses. The most common form of reasoning for this kind of process is abductive

reasoning (Peirce, 1992). In this form of reasoning one seeks the theory that is the simplest and most likely explanation for the observations. Probabilistic abduction is used extensively in areas where it is desirable to compute a quantitative estimate of the likelihood of each possible hypothesis, such as for making diagnoses from medical tests. Some examples of assessment processes include Bayesian networks, Least-Squares Optimization/Regression of solutions for inverse problems, and null hypothesis testing techniques such as Sequential Probability Ratio Tests (SPRT). For an issue tracking system, assessment is typically done by people. This is especially true for bug tracking. Since software is an artifact rather than an object in nature, the developers of the artifact are often the ones most qualified to determine the cause of a bug.

The Resolution subclass represents the processes that transform perceptions and hypotheses into directives. Such a process must usually make decisions under uncertainty. This involves weighting the different outcomes using quantitative criteria such as costs and benefits. There are many techniques for decision making under uncertainty, including influence diagrams, Dempster-Shafer theory, decision trees, and Prognosis of Remaining Useful Life. Many of these technique, such as influence diagrams, can make a decision without the need for determining any underlying causes. In other words, one can combine the assessment and resolution processes. This is shown as an alternative path between Perception and Directive in Figure 2. For the issue tracking example, resolution is typically done by a team that has been given the responsibility for making the decision about how to resolve the issue. For bug tracking, resolution is the process of determining the best way to fix the bug. Unlike issue tracking in general, in bug tracking a single individual may be responsible for fixing a bug.

The Enactment subclass represents the processes that transform reality in some way. Such a process can also produce new facts, thus closing the CARE Loop as shown in Figure 2. Enactment processes involve a wide variety of control structures. An enactment action might be as simple as publishing a new policy or it may be as complex as performing a medical procedure on a patient. For issue tracking systems, the action plan may be encoded in scripts. In the case of bug tracking, such a script would fix the bug by patching the software.

The interpretation of the issue tracking process in terms of the KIDS framework is summarized in Table 1. The example shown in Table 2 is the case of a memory leak caused by a Java program that keeps track of the objects of a class by using a Map. The objects are not garbage collected even after they are no longer needed because they are still in the Map. The correction is to redesign the Map to use weak references that allow the objects to be freed. The steps in the CARE loop are self-explanatory except for the Assessment activity, which consists of proposing various scenarios that could produce the observed behavior, and then selecting the ones that fit most closely. In this example, the scenario about the Map fits much better than any other scenario.

5 The KIDS Ontology

In this section we give an overview of the KIDS ontology, showing selected parts of the ontology. The ontology is represented in OWL. The full KIDS ontology can be found at Baclawski (2016). The Protégé tool Protege was used for ontology development. As

Table 1: The Issue Tracking Process

KIDS Class	Issue Tracking Concept
Fact	Data provided by the participant who raised the issue
Relevance	Process of filtering data relevant to the issue, an activity usually performed by the customer
Classification	Classification of the issue
Perception	Issue category and properties
Assessment	Process whereby the cause of the issue is determined
Hypothesis	Possible causes of the issue
Resolution	Selection of a plan to deal with the issue
Assessment/ Resolution	Selection of a plan to deal with symptoms without determining the actual cause
Directive	Plan for addressing the issue
Enactment	Actions that are taken to try to resolve the issue

Table 2: Issue tracking example for a memory leak in a Java program

KIDS Class	Example
Fact	stack traces, heap dumps
Relevance	only heap dumps are relevant
Classification	perform statistical analysis
Perception	class with largest memory use
Assessment	compare with various scenarios
Hypothesis	memory leak due to a Map
Resolution	redesign to free objects
Assessment/Directive	not used in this scenario
Enactment	participants implement bug fix

Table 3: Ontology Prefixes

Prefix	Ontology
kids:	KIDS Ontology
prov:	PROV-O Provenance Ontology
icom_core:	ICOM Core Ontology

already noted in Section 2, the KIDS ontology makes use of many other ontologies. The ontology from which a concept arises is specified with a prefix. The prefixes are shown in Table 3.

5.1 Data and Reasoning

The Devlin situation theory represents data using infons, the class of which is **kids:Infon**. A situation is an instance of **kids:Situation**, which has the subclasses **kids:ResourceSituation**, **kids:UtteranceSituation** and **kids:FocalSituation**. The classes **kids:Infon** and **kids:Situation** are subclasses of **owl:Thing**, which allows infons and situations to be used as attribute values of other infons. The **kids:supportsInfon** property specifies the infons that a situation supports.

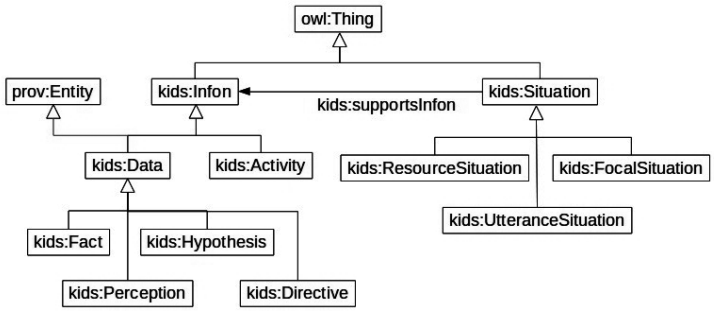

Figure 3: Diagram of the Data hierarchy of the KIDS Ontology along with some of the related classes from the PROV-O and STO ontologies

In the KIDS ontology, there are two kinds of infons: data infons and activity infons. The class of data infons is **kids:Data**. As described in Section 4 above, KIDS subclassifies the **kids:Data** class into four subclasses depending on the mode of reasoning being employed. These are **kids:Fact**, **kids:Perception**, **kids:Hypothesis** and **kids:Directive**, see Figure 3. The **kids:Data** class is a subclass of **kids:Infon** because data is represented using infons. In addition, **kids:Data** is a subclass of **prov:Entity** from the PROV-O ontology, which allows data infons to be annotated with provenance information.

Activity infons are more complex in the KIDS ontology. Each activity is performing a reasoning process using data supported by an input situation and producing data that is marked as being supported by an output situation. The class

of reasoning processes is called **kids:Reasoning**. KIDS classifies reasoning processes into subclasses depending on the stages of the decision making process. These are **kids:Relevance**, **kids:Classification**, **kids:Assessment**, **kids:Resolution**, **kids:AssessmentResolution**, and **kids:Enactment**. An invocation of a reasoning process is an instance of **prov:Activity**. Each activity should invoke exactly one reasoning process. The object property that specifies the reasoning process invoked by an activity is **kids:invokes**. The **kids:invokes** property is an **owl:ObjectProperty** and an **owl:FunctionalProperty**, whose domain is **prov:Activity**. This only ensures that each activity invokes at most one object. The remaining constraints on activity invocation are specified for each subclass of **prov:Activity**. Accordingly, there is one subclass of **prov:Activity** for each subclass of **kids:Reasoning**; namely, **kids:Filter**, **kids:Classify**, **kids:Assess**, **kids:Resolve**, **kids:AssessResolve**, and **kids:Enact**, respectively. The subclasses of **prov:Activity** are actually defined by OWL class constructors. For example,

$$\text{kids:Classify} \equiv \exists \text{kids:invokes.kids:Classification}$$

This class constructor ensures that each **kids:Classify** activity invokes at least one reasoning process and that the activity invokes a **kids:Classification** reasoning process. When combined with the previous constraints on **kids:invokes**, each **kids:Classify** activity invokes exactly one **kids:Classification** reasoning process. Therefore, one can infer the following:

$$\text{kids:Classify} \equiv \exists \text{kids:invokes.kids:Classification} \sqcap \forall \text{kids:invokes.kids:Classification}$$

Those classes that are defined by class constructors are italicized in the class diagrams.

In general, several activities can be used for one step in decision making. The activities performed by one step form an utterance situation. Consequently, one can derive six subclasses of **kids:UtteranceSituation** depending on what kind of activities it supports: **kids:RelevanceSituation**, **kids:ClassificationSituation**, **kids:AssessmentSituation**, **kids:ResolutionSituation**, **kids:AssessmentResolution Situation**, and **kids:EnactmentSituation**, respectively. These situation classes are derived classes defined using class constructors. For example,

$$\text{kids:AssessmentSituation} \equiv \text{kids:UtteranceSituation} \sqcap \forall \text{kids:supportsInfon.kids:Assess}$$

The reasoning and activity hierarchies are shown in Figure 4.

As we described in Section 4 above, each reasoning process defines the requirements for its inputs, the **kids:inputSpecification** of the process, and specifies the characteristics of its outputs, the **kids:outputSpecification** of the process. Each of these is expressed as a **kids:Guard**. A specific implementation of a **kids:Guard** is described in Section 6.2. Each reasoning process is annotated using PROV-O with the skills and capabilities provided by the process in the decision making process.

5.2 Capturing the Decision Making Process

A decision making process begins with background data obtained by observations. The background data is represented as infons supported by a **kids:ResourceSituation**. To

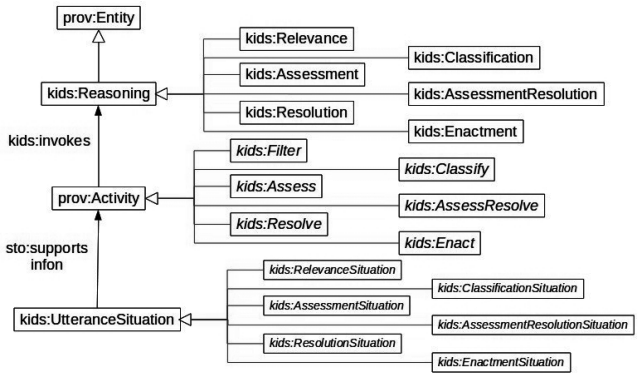

Figure 4: Diagram of the Reasoning, Activity and Utterance Situation hierarchies of the KIDS Ontology and the relationships between the hierarchies

start a CARE Loop, relevance reasoning is performed to select the infons in the first **kids:FactSituation** from the infons in the **kids:ResourceSituation**. The purpose of the decision making process is to achieve an explicitly stated goal. The goal determines the relations and objects that are relevant. This is done by means of a **kids:Guard** object. The clauses in the **kids:Guard** are used to filter the relevant infons from the **kids:ResourceSituation** to compose a **kids:FactSituation**, which is a subclass of **kids:FocalSituation**. For added performance, one can also restrict attention to relevant data that has changed significantly since a previous decision making cycle. However, even if the ultimate goal is to answer a query involving a single class and property for a single object, other classes, properties and objects may also be relevant because of rules and other reasoning processes. Consequently, a relevance reasoning step will often be necessary. Some techniques for relevance reasoning are discussed in more detail in Section 6.2 below. Extracting the infons having a relevant relation (i.e., relevant class or property in the terminology of OWL) is done with T-box reasoning. Extracting the infons having a relevant object is done with A-box reasoning. Not all CARE loops need a relevance reasoning step. In an issue tracking system, the data provided for an issue has already been selected for relevance by the agent who raised the issue.

Having constructed a **kids:FactSituation**, one can now start the decision making process. This is a loop that can proceed as follows: the **kids:FactSituation** is used to produce a **kids:PerceptionSituation**, which is used to produce a **kids:HypothesisSituation**, which is used to produce a **kids:DirectiveSituation**, which is used to produce a **kids:FactSituation**, and the process then proceeds iteratively. Many variations are possible. For example, one can skip producing a **kids:HypothesisSituation**, directly producing a **kids:DirectiveSituation** from a **kids:PerceptionSituation**. One can also explore alternative (sequential) or concurrent (parallel) branches from a single input situation. This allows one to use a variety of execution models. While the actual execution model is outside the scope of the KIDS ontology, one can use annotations to record how the decision making process is being executed. The situation that is used as input to the utterance situation and the situations

that are produced as output from the utterance situation are connected by object properties between **kids:UtteranceSituation** and **kids:FocalSituation**. The object properties are **kids:usesSituation** and **kids:producesSituation**. The situations (activities and data) in a **kids:CARE-Loop** form a directed graph via the **kids:usesSituation** and **kids:producesSituation** object properties. At the activity level, the infons that are used as input to an activity are specified by **kids:usesInfon**, and the infons that are produced as output from an activity are specified by **kids:producesInfon**. The **kids:partOfLoop** object property specifies the CARE Loop to which a situation belongs.

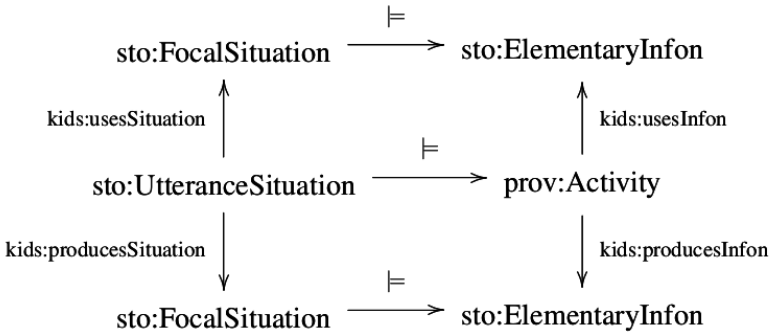

Figure 5: Diagram of uses, produces and support properties interpreted as morphisms in the category of binary relations

The various uses, produces and support properties form the diagram in Figure 5. The arrows in this diagram are morphisms in the category[2] of binary relations, in which the composition of morphisms is defined by joining them. The upper square in the diagram above is not necessarily commutative because an activity need not use every infon in the focal situation. However, the lower square should be commutative as this ensures that if an utterance situation u produces a focal situation s, then the infons supported by s are exactly the ones that are generated by activities supported by u. The commutativity of the lower square is enforced in the KIDS ontology by means of the OWL 2 property chain construct (Hitzler, Krötzsch, Parsia, Patel-Schneider, & Rudolph, 2009, Section 6.2).

In the most general CARE Loop, a **kids:Classification** process transforms **kids:Fact** to **kids:Perception**; a **kids:Assessment** process transforms **kids:Perception** to **kids:Hypothesis**; a **kids:Resolution** process transforms **kids:Hypothesis** to **kids:Directive**; and a **kids:Enactment** process transforms **kids:Directive** to **kids:Fact**. The last transformation is more complex than the others. It is depicted in Figure 6 and is explained in more detail below. However, in more specialized CARE loops, one can have feedback loops, and one can even skip over a step in the general loop. For example, one can skip constructing an hypothesis, and go directly from **kids:Perception** to **kids:Directive** with a process in **kids:AssessmentResolution**

[2]The word "category" in this case is from mathematical category theory. It was used here only as a means of showing how various properties are constrained in the KIDS ontology. The actual constraints are specified using OWL property chains.

which is a transformation that combines assessment and resolution. This is commonly done in an issue tracking system where one can initially propose a solution that deals with the problem, effectively masking it, without determining the underlying cause. An example is given in Section 6.2.

5.3 Bitemporality

An activity has two times: the transaction time and the valid time. The transaction time is the time when the actual processing of the activity takes place. The valid time is the time interval during which the activity is meaningful within a situation. The valid time can be either in the past or the future relative to the transaction time. By having two notions of time, one can deal with the conflicting requirements of the actual processing operations on computing hardware and the time in the world when events take place. This split between transaction time and valid time is called *bitemporality* and is a fundamental part of the KIDS framework. The transaction time is represented in the KIDS ontology by the provenance annotations **prov:startedAtTime** and **prov:endedAtTime** as these are concerned with the activity that was performed. Valid time, however, is not about when the activity occurred but when rather the period of time in the situation during which data are valid. This concept of time is represented by **kids:hasValidTime**, whose value is in **kids:TimeInterval**. The starting time for a time interval is specified with **kids:startTime**, and the ending time with **kids:endTime**. The ending time can be either an actual time or some reason why the ending time is not specified, such as that the period of time proceeds indefinitely into the future, or that the period of time is instantaneous so the ending time is not applicable.

Bitemporality gives rise to two notions of sequence: a succession in valid time and a succession in transaction time. We use the object properties **kids:followsInValidTime** and **kids:followsInTransactionTime** to represent the next **kids:Situation** in each of the sequences. These properties differ from **prov:wasDerivedFrom** because the next situation is not derived from the previous one in the sequence. However, these properties are stronger than simply that the situations occur in a linear sequence of times, because they may be regarded as being part of a single larger situation. For example, when one has a series of medical tests this would be represented with **kids:followsIn ValidTime** because the results of the tests are not necessarily available immediately or in order by transaction time. On the other hand, if one is performing a series of database queries and updates, then this would be represented with **kids:followsInTransaction Time**. In practice, both kinds of sequences can occur in a single CARE loop.

5.4 The CARE Loop within a Context

The purpose of a decision making process is to modify the world to achieve a goal on behalf of an agent. The modification of the world is itself a complex activity which we have not explicitly modeled. In Figure 6, we show how the CARE Loop interacts with the world, although the ontology does not model the sensors, actuators, embedded controllers and the other agents affected by the decision making process. It is assumed that the agent that is running the CARE Loop is responsible for either directly implementing the directive activities or ensuring that they are carried out by other agents. When other

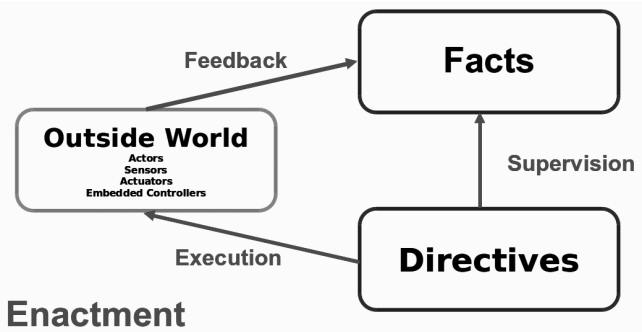

Figure 6: Diagram illustrating how the CARE Loop of the KIDS Framework interacts with the World

agents are carrying out the directive activities, the agent acts as a supervisor, which is represented by the Supervision link in Figure 6. For example, if a directive is not completed in a timely fashion, then the agent must intervene or raise an exception (if the other agents are software systems), and attempt to perform corrective actions. In the bug tracking example, directives are plans for correcting a customer issue. The supervising agent would normally set a time limit for the plan to be completed. If the plan was not completed by the deadline, then the priority of the issue would be escalated so that it receives more attention and resources.

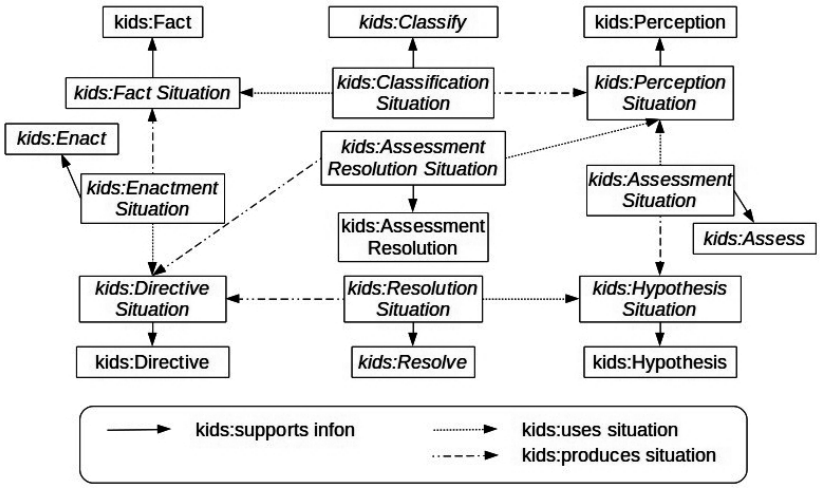

Figure 7: Diagram of the CARE Loop in terms of the KIDS Ontology

The data and activities of the CARE Loop (except for the relevance reasoning step) are shown in Figure 7. For example, the assessment step is performed by activities in the **kids:Assess** class. Such an activity uses a **kids:PerceptionSituation** and produces a **kids:HypothesisSituation**. This diagram is the ontological representation of the

decision making process that is depicted informally in Figure 1 and semi-formally in Figure 2.

Each CARE Loop is invoked by a **prov:Agent**, which can be a person, organization or software process. The **prov:actsOnBehalfOf** property specifies the agent that is responsible for invoking the CARE Loop. While the agent who invokes the **kids:CARE-Loop** is ultimately responsible for the activities that are performed, the invocations themselves are often the responsibility of some intermediary, usually a software process. The software process is represented as a **prov:SoftwareAgent**. To allow a CARE Loop to be part of a collaborative activity by multiple individuals, it can have an associated **icom_core:Space**. This is specified with the **kids:hasWorkspace** property. The ICOM ontology models a large variety of communication and collaboration mechanisms, including email, chat, teleconferences, wikis, blogs, and social networking. ICOM is a modular ontology with twelve modules, including access control (icom_ac module), document management (icom_doc module), email and chat (icom_msg module), and attachments in other media (icom_content module). Collaboration is especially important for an issue tracking system because issue resolution can involve large geographically distributed teams, such as for open-source projects.

5.5 Comparison of the KIDS Ontology with the KIDS Model

Although the KIDS ontology developed in this section is intended to be the ontology for the KIDS model developed in Gawlick et al.; Liu et al.; Chan et al., there are a number of differences. While the concept of "situation" is similar in both the KIDS model and the KIDS ontology, in the ontology we used the STO concept rather than the concept in the KIDS model. In particular, an STO situation is a flexible concept that allows situations to contain other situations and also distinguish resource, utterance and focal situations, none of which were developed in the KIDS model.

Another difference between the KIDS model and the KIDS ontology is the representation of the execution history. The KIDS model includes a loop counter, which represents discrete time T_k with an index $k = 1, 2, \ldots n$, where n is the current loop counter of a CARE Loop. The model identifies an activity in a CARE Loop by a time index k, e.g., CARE-Loop$_k$.classify, CARE-Loop$_k$.assess, CARE-Loop$_k$.resolve, and CARE-Loop$_k$.enact. The model stipulates a function that maps the discrete time index to the valid times. The activities for the same index k have the same valid time, which normally advances monotonically with index k such that the valid time of a CARE loop at time index k is normally $[T_k, T_{k+1})$, where $T_{k+1} > T_k$. However, the model allows for out of order execution of activities with respect to the valid times, i.e., $T_{k+1} < T_k$ for some k, and so the valid time of a CARE loop at time index k is generally $[T_k, min\{T_j \mid j < n, j \neq k, T_j > T_k\})$ where n is the current loop counter. The model also stipulates that the time index can be reset to a lower number in order to re-evaluate past situations and activities. By contrast, the KIDS ontology is much more flexible by specifying the execution history implicitly in the directed graph of situations and activities. The directed graph is more flexible for representing the repeated executions of the activities, by branches in the directed graph. With this flexibility of branching in the directed graph, the KIDS ontology avoids the conceptual difficulty of the KIDS model in representing the resetting of the time index. It also allows for

exploration concurrently in alternative paths as well as backtracking to revisit previous phases.

The KIDS model uses Flexible Schema Data (FSD) and Features to represent data. These are handled in the KIDS ontology by using **kids:Infon**, which was specified to be a **prov:Entity** so that it can be annotated with provenance information. FSDType and FeatureType are available in the KIDS ontology as **kids:Relation** which corresponds to the situation theory concept of relation. Incidentally, while **kids:Infon** was adequate for the use cases, it is likely that a more general notion of infon would be necessary for other application domains.

The KIDS model uses the term "knowledge" to mean a function by which data is processed to produce new data. An example of such a mechanism is a rule, but many other kinds of mechanism can be used. Unfortunately, the term "knowledge" includes information as well as skills, so it is much too general for how it is used in the KIDS model. Consequently, in the KIDS ontology, the word "reasoning" was used instead. The common superclass of data and knowledge is called CARE in the KIDS model. The class corresponding to CARE is **kids:Infon** in the KIDS ontology. The invocation of a knowledge function is an activity, so it was handled in the KIDS ontology using **prov:Activity** from the PROV-O ontology. The collection of all activities performed for one stage of a CARE Loop is handled using a **kids:UtteranceSituation**.

The notions of actor, actor profile, owner, agent, assignee, encoding, description and language of the KIDS model were handled in the KIDS ontology by the PROV-O ontology and XML Schema Datatypes. The KIDS model includes a notion of vector for representing data as well as a figure of merit. Since vectors as well as many other data structures can be represented using relations, the notion of **kids:Infon** is sufficient for representing data. The notion of an entity in the KIDS model was handled in the KIDS ontology using **prov:Entity**.

6 Data Diversity and Execution Models

Traditional systems, including relational databases (Codd, 1970) and ontology-based systems, presume that the schema or ontology is developed prior to storing any data. Not all data is effectively handled in this manner, such as documents and spreadsheets that do not have a fixed schema or ontology. We call this kind of data Flexible Schema Data (FSD). This refers to schema-less unstructured document content and semi-structured data, as well as structured data such as linked data (Bizer, Heath, & Berners-Lee, 2009) whose metadata requires continuous evolution. It is a common practice for users to process such data with systems that do not require up-front schema modeling and ontology development to store and process data with these characteristics. In this section we first discuss the issues that arise with FSD, then propose detailed execution models for processing FSD occurring in a decision making process, and finally discuss how KIDS can orchestrate the processing of FSD using some Big Data tools.

6.1 Flexible Data Issues

Data that we refer to as FSD are already implemented in many systems. Content management systems are used for document, image and video storage, and such systems build a text index to allow users to search their content efficiently (Salton & McGill, 1986). Semi-structured data is commonly represented in XML and increasingly in JSON. Native database systems have been developed for storing such data, and new systems are continually emerging as well. XML can be queried with XQuery. While there do exist query languages for JSON, they are relatively recent, with a great variety of syntaxes and semantics for them. NoSQL systems are frequently used for semi-structured data, and there is a trend toward using JSON as the preferred format for the NoSQL community. However, there is little commonality among all of the formats and query languages for this great variety of unstructured and semi-structured data. As a result, management of data is an increasingly complex task. Users must use many data platforms that have incompatible capabilities and query languages. Integration of such data is currently handled in an ad hoc manner by application programmers.

To deal with this problem, database management systems, especially relational database management systems (RDBMS), are being extended to be able to manage a variety of datatypes beyond the traditional relational data model. The SQL language has been extended to include support for user-defined datatypes (UDT) as well as hierarchically structured data such as XML and JSON. This approach leverages the advantages and lessons learned by the RDBMS community. However, this approach still requires the development of a schema prior to storage and retrieval of data. Liu and Gawlick have proposed three principles to extend RDBMS platforms to manage FSD in the new extended relational systems; namely, principles for data storage, querying and indexing. See Liu and Gawlick (2015) for details.

For the issue tracking example, issues can be raised that use completely new kinds of data, such as unstructured documents, images and videos. When such issues are raised, they introduce their own metadata. Videos, for example, have rendition metadata such as frame rates and formats. Such metadata would be outside the existing schema for the issue tracking system. An issue tracking system would normally have no mechanism for dealing with such data and metadata as anything but unstructured data.

6.2 Processing Flexible Data

The most important difference between FSD and RDBMS is the decentralization of metadata in FSD compared with the centralized schema requirement of an RDBMS. We propose that ontologies are well suited for managing such decentralized metadata. Ontologies based on RDF/S and OWL use open-world semantics which allows for incorporating new metadata as new forms of data are being stored in the system. However, processing OWL ontologies can be very inefficient because of the computational complexity of description logic. To deal with this problem, we use reasoning processes which use one situation to produce another situation. Such reasoning processes can be non-monotonic. For example, one can use statistical techniques with Bayesian networks and influence diagrams in **kids:Assessment** and **kids:Resolution** reasoning pro-

cesses. To reason about relevant relations and individuals of **kids:RelevanceSituation**, one can use database technologies such as registered, bitemporal, flashback queries, and expression filters as described below.

As discussed in Section 5.2, a **kids:Guard** is responsible for selecting the relevant relations and objects for a reasoning activity. The **kids:Guard** binds the variables of the **kids:inputSpecification** and **kids:outputSpecification** of a **kids:Reasoning** process with the objects in a CARE Loop to formulate a query to filter the relevant infons from a situation. The results of this query are used to compose another situation in the CARE Loop. The query is registered with the database where the expressions in the query are indexed using expression filters (stored conditional expressions) (Yalamanchi, Srinivasan, & Gawlick, 2003). The expression filters can be used to filter the expressions that are relevant to the new infons, whose polarity can change in either direction so that they can either assert or retract infons to enable non-monotonic reasoning with situations. The relevance reasoning involves mutual filtering; namely, while the guard clauses are filtering the relevant infons, the expression filters are filtering the guard clauses using new infons. The execution is data-driven: when a **prov:Activity** produces new infons, the expression filters select the guard clauses that are relevant to the new infons, and the selected guard queries are executed. In other words, the execution model is necessarily asynchronous, although if the data is available synchronously, then the processing can also be synchronous. The guard queries need to be executed using flashback query technology, a technology that views database objects in the past. The particular technology we use is Oracle flashback query. Typically, a **kids:FocalSituation** is composed by a flashback query as of the transaction time given by the **prov:endedAtTime** property of the **prov:Activity** that produces the new infons. The **prov:endedAtTime** property of the **prov:Activity** that produces the new infons for the **kids:FocalSituation** is equated to the **prov:startedAtTime** property of the new **prov:Activity** that uses the infons of the **kids:FocalSituation**. On the other hand, the **prov:endedAtTime** property of the new **prov:Activity** may be represented by the log sequence number generated when its transaction is committed.

Typically a **kids:Reasoning** process invoked by a **prov:Activity** is reevaluated if the **kids:FocalSituation** used by the **kids:UtteranceSituation** that supports the **prov:Activity** changes significantly. This involves querying the two states of the **kids:FocalSituation** by flashback queries as of two different transaction times. This allows one to compare the state of the **kids:FocalSituation** in the past with their current state without having to save a snapshot of each **kids:FocalSituation** used by any **kids:UtteranceSituation**. When a **kids:Guard** is selected by the expression filters, the KIDS engine can execute a first flashback query as of the transaction time given by the **prov:startedAtTime** property of a **prov:Activity** that invokes a **kids:Reasoning** process, which in turn is associated with the **kids:Guard** of the CARE Loop. It compares the result of the first flashback query with the result of a second flashback query as of the transaction time given by the **prov:endedAtTime** property of a recent **prov:Activity** in a different part of the CARE-Loop or a different CARE-Loop. If the two flashback queries show substantive, relevant changes in the **kids:ResourceSituation**, the KIDS engine can reevaluate the **kids:Reasoning** process in a new **prov:Activity** using a new **kids:FocalSituation**; the transaction time of the second flashback query is set as the **prov:startedAtTime** property of the new

prov:Activity. If more than one **prov:Activity** can be invoked in a CARE Loop, the KIDS engine shall invoke them in the precedence ordering: classify $<$ assess $<$ resolve $<$ enact.

The reasoning processes invoked by **kids:Classify**, **kids:Assess**, and **kids:Resolve** activities can be reevaluated any number of times, in the precedence order, as long as the **kids:Enact** activity has not been invoked. While each reevaluation logically produces a new **kids:FocalSituation**, it is more intuitive as well as more efficient to regard the **kids:FocalSituation** as being updated. After the **kids:Enact** activity is invoked, it may require compensating directives to re-invoke the **kids:Enact** activity. Sometimes after a **kids:Enact** activity is invoked, its preceding **kids:Classify**, **kids:Assess**, and **kids:Resolve** activities may be reevaluated to see how the decision process will have performed in the light of the **kids:Fact**, **kids:Perception**, **kids:Hypothesis**, and **kids:Directive** that are updated retroactively to the valid times of the past activities. Like the **kids:Data**, a **kids:Reasoning** and **kids:Guard** can be updated, and their valid times set retroactively to the valid times of the past activity. These updates will trigger the reevaluation of the reasoning processes for **kids:Classify**, **kids:Assess**, and **kids:Resolve** activities. Sometimes the **kids:Enact** activity can be repeated with the compensating directives.

This kind of behavior is important in an issue tracking system. For the issue tracking system example, a common way to deal with issues is to first suggest a temporary solution without determining the actual cause of the problem, and then later determine the underlying cause and solve it. For example, a program might have a bug that results in a memory leak. The users could deal with the problem by increasing memory limits. Later, when the bug has been found and corrected, the users should install the bug correction and "undo" the previous solution (i.e., increasing memory limits).

6.3 Processing Big Data

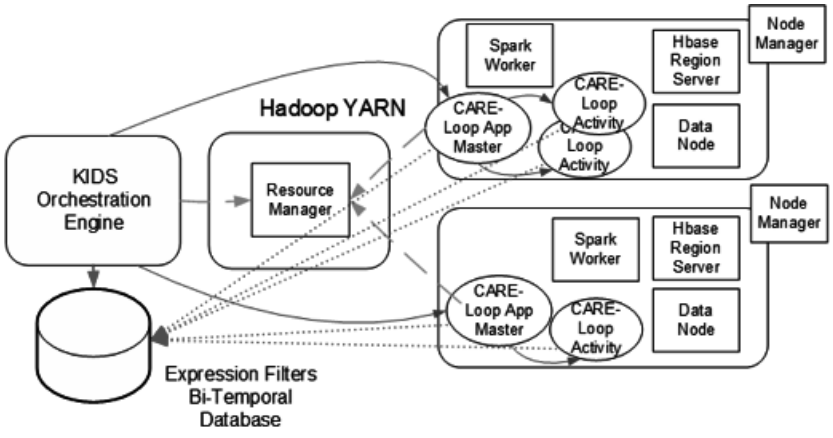

Figure 8: Architecture diagram illustrating how the KIDS Framework can be used to orchestrate Big Data discovery and hypothesis testing processes

An example of how KIDS can integrate inference engines in a Big Data system is shown in Figure 8. The KIDS orchestration engine in Figure 8 can be implemented with Oracle Database technologies, including the registered, bitemporal, flashback queries, and expression filters. The KIDS engine can submit a CARE-Loop application to the Apache Hadoop YARN Resource Manager, which will launch the CARE-Loop Application Master (AM) in a container in one of the compute nodes in the Hadoop cluster. These actions are shown as arrows from the KIDS engine in Figure 8. Each CARE-Loop AM can be a long running process within its node, which reports its activities to the Resource Manager. This is a dashed arrow in the figure from each CARE-Loop AM. The Resource Manager allows the KIDS engine to monitor the status of each CARE-Loop AM. This is shown as a dashed arrow in the figure from the KIDS engine to the Resource Manager. The CARE-Loop AMs and their activities can store the high-value perceptions, hypotheses, and directives as data in the bitemporal database. This is shown as dotted arrows in the figure. The reasoning processes of the CARE-Loop activities can be executed in Apache Spark, can access the data in Apache Hbase, and can take advantage of data locality among the Spark worker nodes, HBase region servers, and Apache Hadoop HDFS data nodes.

Returning to the issue tracking system example, by specifying the metadata for new issues in a standard language, such as OWL, it can be stored, queried and indexed. More significantly, it can be used for integrating data from other relevant issues. The data can be distributed and processed using a Big Data system such as the one shown above. One constructs the situation by combining relevant information from the current issue as well as other issues. Relevance is determined by the intended goal to be achieved. For a bug tracking system, the goal is usually to fix some perceived defect. The data relevant to this goal is gathered together and expressed as the initial **kids:FactSituation**. One can then start a CARE Loop for dealing with the bug in a node as discussed above.

7 Use Cases for the KIDS Framework

We now present some use cases for the KIDS Framework, in four domains that differ very much from each other.

7.1 Customer Support

Customer support is a broad area characterized by problem solving activities for issues raised by the customers of a company. Issue tracking systems form an important part of customer support, although many other activities also come under the general purview of customer support, such as product sales and upgrades. In this section, we focus on issue tracking and resolution, the running example introduced in Section 3.

Obviously, rapid response to customer issues is an important goal of any company. Maximizing the automation of handling known issues whenever economically possible will not only improve issue response time but will also free support and customer personnel to focus on newly emerging issues that require a great deal of collective human experience and intelligence. That is why applications developed in this domain are in

constant flux, due to the never ending demands for automation. Three challenges arise from such automation demands: 1) achieving economical automation, 2) designing the application to enable the rapid deployment of such automation, and 3) achieving precise articulation and provenance of knowledge gained from resolved issues.

As with many problems, attempting to automate everything at once is much too difficult and likely to fail. Accordingly, it is essential that the problem be divided into smaller phases. This is especially important for customer support where the different stages of problem solving have very different levels of complexity. Automating data collection is much simpler than parsing the collected data. Parsing, in turn, is easier than automating diagnosis. Finally, automating remediation is the most difficult of all. This suggests that the overall problem be divided into the four phases just described. However, given the complexity of each of these phases, each of them, especially the later phases will need to be further subdivided. Since each of the major phases is itself a decision making process, this leads to the overall decision making process shown in Figure 9.

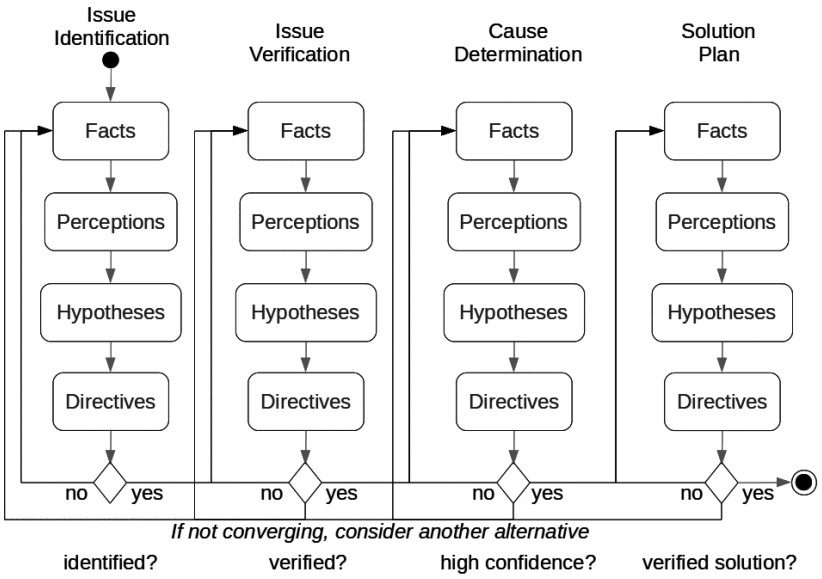

Figure 9: Diagram of the Customer Service decision making process, showing how the steps in a decision making process can themselves be decision making processes

Each of the phases in Figure 9 represents an important kind of decision that must be made in the process. Each decision point has three choices. If a phase is successful, the next phase begins (or for the last phase, the process completes). If a phase is unsuccessful, it is repeated, but if too many repetitions occur, the process goes back to one of the previous phases. Note that the overall process is similar to the CARE Loop: Issue Identification is analogous to Enactment, Issue Verification is analogous to Classification, Cause Determination is analogous to Assessment, and Solution Plan is analogous to Resolution. When the overall process is regarded as a CARE Loop, an

activity need not immediately continue to the next activity but rather may repeat. This is an example of the kind of branching that was incorporated into the KIDS ontology. Another aspect of the Customer Service decision making process is the hierarchical decomposition in which each activity is itself another CARE Loop.

7.2 Healthcare

Patient care is a demanding task driven by data, observations, knowledge, and procedures; all growing in amount and complexity at an ever increasing pace as new medical sensors are developed and deployed. Now that patients are wearing an increasing number of sensors, practitioners will be continuously responsible for all of their patients whether the patients visit the doctor's office or not. So it is no surprise that situation awareness is at the center of patient care. While the collection of Electronic Medical Records (EMR) – a mix of data describing measurements, images (and reads), observations, diagnosis, and treatments – is extremely important, these records are by no means sufficient. EMRs neither organize data into meaningful categories nor describe who has examined the data, when the data were examined, which diagnosis was derived, and how it was derived. When combined with increasing cost pressures and the potential for malpractice lawsuits, the need for better records is even more compelling.

Practitioners use many methods to care for patients; some of the frequently used terms are: evidence based medicine, standard of care, differential diagnosis, and personalized/precision medicine. There are numerous applications supporting healthcare practitioners; however, they capture only a small part of their tasks; furthermore, they are proprietary and opaque, making major extensions, personalization, and fast evolution practically impossible. What is required is a system that helps practitioners in all phases of the treatment, including allowing practitioners to communicate with this system in their language, transforming measurements into a compact form, providing rigorous provenance, alerting practitioners about abnormalities, allowing major extensions and personalization, and supporting evolution of the system.

As with the customer service use case in Section 7.1 above, the decision making process is too complex to be handled all at once, and even the various phases will themselves be complex decision making processes. In the following we will analyze Classification activities from the point of view of the KIDS ontology with its CARE Loop.

Patient care starts with the gathering of evidence, which comes in the form of observed facts. Facts consist of measurements such as lab results, various kinds of images, and bedside monitor waveforms. There are over 700 kinds of lab test, over 400 of which are quantitative (MIMIC, 2016). Using a large variety of classification techniques, observations are transformed into perceptions. For example, sensor measurements are transformed into how much they deviate from the norm. Facts are complemented by clinician observations which are treated as perceptions. The perceptions will be assessed to arrive at one or more diagnoses (hypotheses) determining the root cause of the observations along with an associated confidence. The next step is a treatment plan resulting in directives. A directive can be a specific treatment and/or more tests based on the standard of care. The enactment of the directives will create more facts and the cycle starts again until the hypotheses show that the target has been reached or nothing

should or can be done any more. Most of these steps are and will increasingly be computer supported, allowing the permanent supervision of patients based on a constant stream of facts, perceptions, diagnoses and directives. New computerized knowledge has to be used immediately once it has been verified and released. Any support has to be considered in the context of personalization; the application of knowledge has to be based on preferences of individuals or teams (Personalized Medicine Coalition, 2014).

Consider the classification activity that transforms facts into perceptions. With over 400 kinds of lab test as well as many other kinds of observation, this is already a daunting task. The various observations have many dimensions and ranges, and the ranges often depend on the situation of the patient. Therefore, determining the conditions for notification of a serious or critical condition is a very demanding task. Here is an example: the risk of cardiac arrest can be seen in a blood test hours ahead (Guerra, Gawlick, Bizarro, & Gawlick, 2011). Unfortunately, this requires understanding a complex interrelation between at least ten measurement values, which is difficult even for the best expert and is often not doable at all. Yet computers are very good at this. There is a real need to integrate the results of computer models like this example with the clinical decision making process performed by practitioners. The KIDS framework for ontology-driven decision making can contribute to this need by classifying facts and using the same qualitative classification for many types of facts that practitioners use; for example, normal, guarded, serious and critical. The goal is to use relatively few classifications to help practitioners to not only take advantage of complex computer models but also achieve situation awareness.

7.3 Cloud Services

The premise of cloud computing is that one can achieve economies of scale by pooling physical resources to provide virtually unlimited resources. This is only possible by dynamically allocating resources to meet customer needs. Accordingly, situation awareness is essential, not only for the cloud service as a whole but also for individual customers. Customers have service level agreements (SLAs) with the cloud service provider that must be met. Conformance to SLAs without excessive resource consumption is therefore one of the critical requirements of cloud operations. Cloud systems are often multitenanted, where a tenant is a group of users who share a software instance, each of which has specific privileges; but in other cloud systems, a tenant is a single user who has exclusive access to a virtual system. These operations have a need for continuous monitoring of key performance metrics so that impending SLA violations can be predicted and avoided or quickly resolved. A typical cloud operation has to monitor, diagnose, and manage millions of hardware and software components of the data centers, networks, server machines, virtual machines, operating systems, databases, middleware, applications, etc., in the operator's and tenants' private, public, and hybrid clouds. Traditional IT techniques not only are far too labor intensive for such an environment but are also insufficiently responsive. Obviously, such techniques cannot scale up to the cloud. Cloud services require continuous measurement of important vital signs, time-series analytics, MSET models (cf., Section 4.2, system response models, predictive anomaly detection, classification based on machine

learning, automatic diagnosis and prognosis, decision support, and control capability through automated decision making processes.

Cloud operation requires Big Data systems to handle the volume of machine data carrying the important vital signs for system health, but sensor or data fusion through Big Data is inadequate. Minimally, it needs a framework like real-time complex event processing (CEP) to perceive the situations involving certain relations among a family of entities and correlations of vital measures of the entities to support decision making. The KIDS framework can collect data relevant to one tenant from different points in time into a situation that can be used for rapid, automated decision making. Decisions will result in requests to acquire or to release resources as well as providing warnings about real or potential problems. Using an orchestration architecture such as the one shown earlier in Figure 8, KIDS provides an effective framework for the situation awareness needed for managing resource allocation, detecting anomalies and diagnosing the causes of the anomalies in large scale cloud operations.

The above requirements for predictive analytics call for the framework to enable information fusion across various special types of perceptions, such as the observation, objective, prediction, simulation, seasonal forecast, etc., deduced by different classification processes and different sources of data for related entities. However, Big Data analytics and real-time CEP technologies have remained largely disconnected. It requires a framework like KIDS to integrate them and to augment them with other essential technologies for large scale state management, such as bitemporal databases, expression filters, registered queries, and forward-chaining and backward-chaining orchestration engines to integrate inference engines such as Rete networks (Forgy, 1982), Bayesian networks, MSET, Support Vector Machines, Neural Networks, OWL reasoners, Online Analytical Processing, and miscellaneous time-series algorithms Chan et al..

7.4 Internet of Things

The Internet of Things (IoT) represents a generic class of applications that could potentially benefit from the KIDS framework. In each of these applications, a large number of facts is collected. These facts need to be represented as compact and intuitive perceptions, which will be used to create hypotheses, and finally directives are created to respond to that situation. Furthermore, the various activities that implement this process continuously evolves. In terms of the KIDS ontology, this means that the utterance situations (process level) are being interpreted as focal situations (data level), and a decision making process must make decisions about how to change the activities being performed in the phases of the CARE Loop.

Rather than discuss IoT applications in general, we will analyze a single IoT application to illustrate how the KIDS ontology-driven decision making framework can be used for IoT problems. The problem is known as No-Trouble-Found (NTF), also called No-Faults-Found (NFF) in the prognostic literature (Accenture Communications & High Tech Solutions, 2016). The NTF problem is that electronic systems and electronic compute-and-control components that are integrated with expensive capital assets in manufacturing, utilities, and transportation, have internal mechanisms that generate an alarm indicating failure. When this happens, the electronic product or compo-

nent may be replaced at a cost to the supplier under warranty replacement or a service contract. The electronic product or component is then returned to the supplier or distributor and tested in a services testing laboratory. Between 25% and 70% of the time, depending upon the segment of the electronics industry, the product or component that was returned operates properly when tested. The cost of replacing products or components that should not have been returned has been estimated as about $2B per year to the industrial sectors of manufacturing, utilities, and transportation. Mitigating the NTF problem is a significant challenge which would have significant financial advantages to many industrial sectors.

For example, in the business-critical enterprise computing industry, NTF rates exceed 20% for "Field Replaceable Units" (FRU), which are internal components and system boards that are swapped upon indication of failure.[3] For military defense electronics, NTF rates are greater than 50%. In the consumer electronics industry, NTF rates for returned products are 68% Accenture Communications & High Tech Solutions. In the commercial airline industry, the "big 3" carriers in the USA all have $100M losses per year due to NTFs in aviation electronic systems. High NTF rates are not only a warranty cost issue for electronics systems and components manufacturers, but high NTF rates also cause additional collateral problems by:

- Diverting the supplier's engineering resources to field issues;
- Increasing FRU sparing levels in FRU stocking depots around the world;
- If a component has a 50% NTF rate, one needs to stock 50% more components in sparing depots;
- Decreasing the stock of components available for new revenue systems;
- Increasing component scrap rates;
- Increasing system repair cycle time; and
- Decreasing customer satisfaction and brand loyalty

Sun Microsystems performed in-depth root-cause analyses of NTFs in electronic systems and found that the leading causes of NTFs in electronic systems and FRUs are the following (Gross & Whisnant, 2009):

- Transient/Intermittent Faults
- Threshold Limits on Noisy Physical Variables
- Sensor Degradation Events
- Human Errors During Testing and Diagnosis

[3]This does not mean that 20% of components in enterprise computers fail; it means that of the "failed" FRUs returned to the systems' manufactures, 20% are discovered to be NTF.

For the rest of this section, we discuss how the KIDS Framework coupled with IoT sensor analyses using pattern recognition could mitigate and avoid NTFs for IoT industrial customers in the manufacturing, utilities, and transportation industries.

The NTF-Mitigation Process expressed as a CARE Loop leveraging IoT telemetry is the following:

- Classify quantitative facts to derive qualitative perceptions
  - IoT transducers provide digitized time series archived as Fact data
  - Advanced statistical machine learning techniques such as MSET are used to classify anomalies in the Fact data
  - Perceived anomalies may or may not be a Failure
- Assess the perceptions to infer the hypotheses
  - Advanced statistical machine learning using null hypothesis testing techniques such as SPRT assesses the anomalies to infer the hypotheses
  - Three hypotheses are tested at each evaluation stage: True Failure, True NTF, Proceed to Next Test
- Resolve the hypotheses to formulate directives
  - If True Failure: Send to scrap/recycling
  - If True NTF: Return to Services' Spare Parts Stock
  - If Unresolved, Proceed to Next Test
- Enact the directives to collect new facts, and
- Repeat the loop for the next test.
- As the CARE Loop above proceeds, it is monitored to learn patterns that can improve real-time field sensing to systemically mitigate NTFs from the installed base.

Application of the KIDS framework and analysis has led to a systematic iterative procedure used in a systems testing laboratory when units are returned from the field under warranty or service-contract returns. This particular use case for driving reductions in NTFs is a good example of a case where there is added ROI by breaking the overall process into an iterative sequence of test evaluations, each with the same hypotheses and with similar information collected and knowledge derived, but with the advantage of proceeding from the quickest, easiest, and cheapest testing, to iterations involving incrementally more detailed pattern recognition challenges (and greater testing time). In Section 7.1 we discussed modular decomposition of the problem space into coherent modular components. For the IoT NTF-mitigation use case, we do temporal decomposition. The benefit, of course, for partitioning the application of KIDS methodology in a linear sequence progressing from cheap to more sophisticated prognostics, is that the unit-under-test (UUT) may be confirmed to be failed/degraded at the

earliest time, and the testing process is then terminated and the UUT can be discarded, recycled, or refurbished. However, the greatest challenge for NTFs across multiple electronics industries is for systems/components that power on, but have subtle or intermittent internal degradation modes that only emerge under a more sophisticated suite of thermal/electrical/performance stress testing.

For units that power on, the simplest testing sequence (which can be conducted on large batches of units in parallel) is cycling in a thermal chamber while powered on, and cycling through the full range of ambient temperatures that the units are qualified for. This simple step is motivated by the recognition that many fault modes for electronic systems are accelerated by or triggered by spatial and temporal thermal gradients. This initial step will quickly eliminate units that are truly broken or have internal thermal-acceleration degradation modes. During this (and subsequent) testing sequences, digitized telemetry from all available internal transducers is continuously recorded into a buffer. Although the telemetry signals from multiple sensors and multiple UUTs would be overwhelming for a human tester to digest, pattern recognition can be used to simplify this complexity by making a binary-hypothesis decision: either the UUT is degraded/failed (in which testing is terminated); or the UUT may be NTF, in which case one should continue to the next test sequence and collect additional telemetry data. Of course, the final "true NTF" determination cannot be made until the conclusion of the final testing sequence.

There are at least two additional test sequences for simple electronic components.[4] In the second sequence, a pre-programmed script is run on the UUT via firmware that stresses the UUT in a manner that maximizes the utilization of any internal ICs, CPUs GPUs, ASICs and memory modules. For this component of the testing the desire is to "stretch the envelope" through the full performance range the unit is expected to experience during normal operation.

In the final test sequence, for units that have shown no evidence of degradation to this point, the unit is tested with a deterministic dynamic stress script (which may or may not be the same stress script used in the previous sequence), and all telemetry signatures from all available IoT sensors are compared to identical telemetry coming from a "golden system" (i.e., a new system for which the best engineering judgment is that it is performing as optimally as possible). Note that the "golden system" that is running the same stress script as the UUT may be running side by side on the same test bench. But more advantageously, the telemetry dynamics from the "golden system" can be pre-archived and the telemetry dynamics from very many UUTs can be analyzed by a machine learning technique such as MSET, using the telemetry dynamics from the "golden system" for the training purposes. This allows parallel testing of large batches of "suspect" NTF units for maximized facility throughput.

In the KIDS CARE Loop for IoT NTF characterization and mitigation, the Agent can be a human testing engineer and would be appropriate if expensive components are tested individually and infrequently. But for large scale parallel testing operations with large batches of components, the KIDS process outlined here is also quite amenable to autonomous agent control as illustrated schematically in Figure 10.

[4] Still more test sequences would be employed for complex computing systems that may contain hundreds of internal transducers.

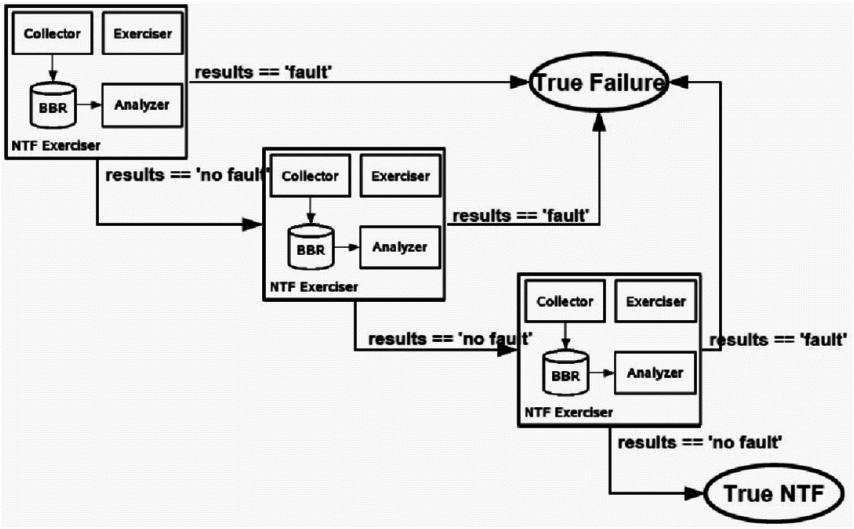

Figure 10: KIDS Sequential Hypothesis Flowchart for Electronic Systems and Components Evaluated as "Suspect NTFs"

8 Conclusion and Future Work

This article has developed a formal, ontology-driven framework for the process of decision making. Our framework may be regarded as an ontology-based formalization of the existing KIDS model, but we have added many new features and solved two of the outstanding problems in the formalization of decision making processes. Like the KIDS model, our ontology classifies both the kinds of data and the reasoning activities that are used to develop situation awareness, but our ontology develops the notion of situation much more thoroughly than in earlier work, aligning it with well-known theoretical foundations for situations. In particular, the classification of situations into focal, utterance and resource situations is a fundamental feature of our ontology. Another new feature of our framework is the flexible process model that supports either synchronous or asynchronous processing and allows the process to explore alternative or concurrent branches. In particular, we developed several notions of time and time series. Relatively detailed descriptions of implementation techniques were presented to make the case that the framework is feasible. The applicability of the framework was illustrated by four use cases from different domains. The four use cases were for customer support, healthcare, cloud service systems and the No-Trouble-Found problem in the Internet of Things. We feel that these use cases give a compelling justification for continued development of our framework.

Although the KIDS model had already been developed as a mathematical model in a series of earlier articles, it was nevertheless a substantial challenge to develop the KIDS ontology. The KIDS ontology introduces many classes that were not explicit in the KIDS model, making distinctions such as subclassing situations, and adding many

axioms not in the KIDS mathematical model. It was also discovered that some base classes in the KIDS model could be constructed using OWL class constructors, and the execution model of the KIDS model could be generalized in a way that has significant benefits for some of the use cases. The integration with the PROV-O PROV-O ontology not only simplified the KIDS ontology by reusing PROV-O, but also allows one to use PROV-O compliant software tools and techniques for maintaining provenance of both data and activities.

As with any ontology engineering process, during the development of the KIDS ontology there were many lessons learned. Unlike many ontology development projects, we had the advantage of having a mathematical model on which to build. This model was based on many use cases as well as the OODA loop. Nevertheless, the development of the KIDS ontology was not straightforward. As already noted, we found that some classes could be derived from others and there were a number of implicit assumptions, now captured as axioms. Integrating KIDS with other ontologies was another challenging activity. While integrating KIDS with PROV-O was relatively easy, our attempt to integrate KIDS with STO encountered a number of obstacles that ultimately prevented us from reusing STO, such as incompatibility with OWL DL reasoners and insufficiently general constructs.

Another issue concerned ICOM and the Rationale Ontology as mentioned in Section 2. ICOM was not fully integrated with the KIDS ontology, and the Rationale Ontology was not integrated at all. The difficulty is that ICOM and the Rationale Ontology have many classes that are similar to those in the KIDS and PROV-O ontologies, yet differ semantically. Bridging between these ontologies remains a significant challenge, but would have significant benefits, such as improved integration with collaborative work environments. As future work, we plan to investigate how to bridge the KIDS ontology with ICOM and the Rationale Ontology.

An even more interesting prospect for future work is to add services for storing and transporting the data and activities of a decision making process, especially the situations. One possible approach for providing these important features is to integrate the KIDS ontology with the Content Management Interoperability Services (CMIS) ontology (Müller & Baclawski, 2015). This would allow one to use one of the large variety of content management systems as the basis for a decision making system.

Acknowledgments

We wish to acknowledge the continuing support of Oracle to the development of the KIDS framework. This work was conducted using the Protégé resource, which is supported by grant GM10331601 from the National Institute of General Medical Sciences of the United States National Institutes of Health. We would also like to thank the referees for many suggestions that improved the quality of our presentation.

This article was published by IOS Press in *Applied Ontology, 12(3-4), 245–273.* The final publication is available at IOS Press through http://dx.doi.org/10.3233/AO-170189

References

Baclawski, K. (2016). *The KIDS Ontology version 2.0.* Retrieved December 9, 2017 from `http://bit.ly/2xZuTNJ`

Baclawski, K., Kokar, M., Matheus, C., Letkowski, J., & Malczewski, M. (2003). Formalization of situation awareness. In H. Kilov & K. Baclawski (Eds.), *Practical foundations of behavioral semantics* (pp. 25–40). Dordrecht, Netherlands: Kluwer Academic.

Baclawski, K., Malczewski, M., Kokar, M., Letkowski, J., & Matheus, C. (2002, November 4). Formalization of situation awareness. In *Eleventh OOPSLA Workshop on Behavioral Semantics* (pp. 1–15). Seattle, WA.

Baclawski, K., Malczewski, M., Kokar, M., Letkowski, J., & Matheus, C. (2006). *The Situation Theory Ontology.* Retrieved December 9, 2017 from `http://bit.ly/1yrikQj`

Baclawski, K., & Thessen, A. (2014, February 13). Tackling the variety problem in Big Data. In *Ontology Summit 2014 Big Data and Semantic Web meet Applied Ontology*. Retrieved December 9, 2017 from `http://bit.ly/2xefFQT`

Barwise, J. (1981). Scenes and other situations. *J. Philosophy*, *77*, 369–397.

Barwise, J. (1989). *The situation in logic* (Vol. 17). Menlo Park, CA: CSLI/SRI International.

Barwise, J., & Perry, J. (1983). *Situations and Attitudes*. Cambridge, MA: MIT Press.

Big Trouble with "No Trouble Found" Returns: Confronting the High Cost of Customer Returns. (2016). Retrieved December 9, 2017 from `http://bit.ly/2vO5QZD`

Bizer, C., Heath, T., & Berners-Lee, T. (2009). Linked data–the story so far. *International Journal on Semantic Web and Information Systems*, *5*(3), 1–22.

Boyd, J. (1976, September 3). *Destruction and creation* (Tech. Rep.). U.S. Army Command and General Staff College. Retrieved December 9, 2017 from `http://bit.ly/1aAje2`

Bruegge, B., & Dutoit, A. (2009). *Object-Oriented Software Engineering using UML, Patterns, and Java third edition*. Prentice Hall.

The case for personalized medicine, fourth edition. (2014). Personalized Medicine Coalition. Retrieved December 9, 2017 from `http://bit.ly/TiMYxZ`

Chan, E., & Baclawski, K. (2013). *Integrated Collaboration Object Model (ICOM) for Interoperable Collaboration Services Version 1.0: Committee Specification 01*. Organization for the Advancement of Structured Information. Retrieved December 9, 2017 from `http://bit.ly/2xYBoR5`

Chan, E., Behrend, A., Gawlick, D., Ghoneimy, A., & Liu, Z. (2012). Towards a synergistic model for managing data, knowledge, processes, and social interaction. In *Proceedings of the Society for Design and Process Science (SDPS).* Berlin.

Chan, E., Gawlick, D., Ghoneimy, A., & Liu, Z. (2014, October 27–30). Situation aware computing for Big Data. In *Workshop on Semantics for Big Data on the Internet of Things (SemBIoT 2014), 2014 IEEE International Conference on Big Data.* Washington DC.

Codd, E. (1970). A relational model of data for large shared data banks. *Commun. ACM*, *13*(6), 377–387.

Darian, S. (2003). *Understanding the language of science*. University of Texas Press.

Devlin, K. (1991). *Logic and Information*. Cambridge, U.K.: Cambridge University Press.

Dreier, A. (2012). *Strategy, planning and litigating to win*. Boston, MA: Conatus Press.

Duggar, V. (2008). *Semantic annotation for decision support* (Unpublished master's thesis). Northeastern University.

Duggar, V., & Baclawski, K. (2007, November 5). Integration of decision analysis in process life-cycle models. In *International workshop on living with uncertainties.* Atlanta, Georgia, USA.

Fioratou, E., Flin, R., Glavin, R., & Patey, R. (2010). Beyond monitoring: distributed situation awareness in anaesthesia. *British journal of anaesthesia*, *105*(1), 83–90.

Forgy, C. (1982). Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artif. Intell.*, *19*, 17–37.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

Gawlick, D., Chan, E., Ghoneimy, A., & Liu, Z. (2015). Mastering situation awareness: The next big challenge? *SIGMOD Record*, *44*(3), 19–24.

Georgakopoulos, D. (2008). Engineering OODA systems: Architectures, applications, and research areas. In *On the move to meaningful internet systems: Otm 2008* (Vol. 5332, pp. 1215–1216). Berlin: Springer-Verlag.

Gross, K., & Whisnant, K. (2009, October 27). *Process for resolving "no trouble found" server products.* United States Patent No. 7,610,173 Assigned to Sun Microsystems, Santa Clara, CA

Guerra, D., Gawlick, U., Bizarro, P., & Gawlick, D. (2011). An integrated data management approach to manage health care data. *Datenbanksysteme für Business, Technologie und Web*, 596–605.

Healthcare, S. (2016). *Situational Awareness in Healthcare.*

Hitzler, P., Krötzsch, M., Parsia, B., Patel-Schneider, P., & Rudolph, S. (2009). *OWL 2 Web Ontology Language Primer.* Retrieved December 9, 2017 from `http://bit.ly/2xYZ1sB`

Klein, G., Moon, B., & Hoffman, R. (2006a). Making sense of sensemaking I: alternative perspectives. *IEEE Intelligent Systems*, *21*(4), 70–73.

Klein, G., Moon, B., & Hoffman, R. (2006b). Making sense of sensemaking II: a macrocognitive model. *IEEE Intelligent Systems*, *21*(5), 88–92.

Kornyshova, E., & Deneckere, R. (2010, November). Decision-making ontology for information system engineering. In *ER 2010 29th International Conference on Conceptual Modeling* (pp. 104–117). Vancouver, Canada: Springer.

Lebo, T., Sahoo, S., & McGuinness, D. (2013, April 30). *PROV Ontology (PROV-O).* Retrieved December 9, 2017 from `http://bit.ly/2xPcx2k`

Liu, Z., Behrend, A., Chan, E., Gawlick, D., & Ghoneimy, A. (2012, July 25-27). KIDS - A model for developing evolutionary database applications. In *Proceedings of the International Conference on Data Technologies and Applications (DATA 2012)* (pp. 129–134). Rome, Italy.

Liu, Z., & Gawlick, D. (2015). *Management of Flexible Schema Data in RDBMSs – Opportunities and Limitations for NoSQL* (Tech. Rep.). Oracle Corporation.

Matheus, C., Kokar, M., & Baclawski, K. (2003, July). A core ontology for situation awareness. In *Proc. Sixth Intern. Conf. on Information Fusion FUSION'03* (pp. 545–552).

Matheus, C., Kokar, M., Baclawski, K., Letkowski, J., Call, C., Hinman, M., ... Boulware, D. (2005a, July 25-29). Lessons learned from developing SAWA: A situation awareness assistant. In *Eighth Int. Conf. Info. Fusion.*

Matheus, C., Kokar, M., Baclawski, K., Letkowski, J., Call, C., Hinman, M., ... Boulware, D. (2005b). SAWA: An assistant for higher-level fusion and situation awareness. In *Proc. SPIE conference on multisensor, multisource information fusion* (Vol. 5813, pp. 75–85).

Medical Information Mart for Intensive Care III (MIMIC-III). (2016). Retrieved December 9, 2017 from `http://bit.ly/1Ry9FqP`

Moran, P. (2008). *Full diagram originally drawn by John Boyd for his briefings on military strategy, fighter pilot strategy, etc.* Wikipedia. Available at `http://bit.ly/2xPB8UQ` License: `http://bit.ly/9sOn0`

Müller, F., & Baclawski, K. (Eds.). (2015). *Content Management Interoperability Services (CMIS) Version 1.1 Plus Errata 01*. Organization for the Advancement of Structured Information. Retrieved December 9, 2017 from `http://bit.ly/12o0eyP`

Osinga, F. (2006). *Science Strategy and War, The Strategic Theory of John Boyd*. Abingdon, UK: Routledge.

Pearl, J. (2000). *Causality: Models, Reasoning and Inference*. Cambridge, UK: Cambridge University Press.

Peirce, C. (1992). *Reasoning and the logic of things*. Cambridge, MA: Harvard University Press.

Protégé website. (2015). Retrieved December 9, 2017 from `http://bit.ly/AASA`

Punning. (2007). Retrieved December 9, 2017 from `http://bit.ly/2xYdBjR`

Richards, C. (2004). *Certain to Win: the Strategy of John Boyd, applied to business*. Xlibris.

Salton, G., & McGill, M. (1986). *Introduction to modern information retrieval*. New York: McGraw-Hill.

Steinberg, A., & Bowman, C. (2001). Revision to the JDL data fusion model. In D. Hall & J. Llinas (Eds.), *Handbook of Multisensor Data Fusion* (pp. 2-1–2-19). CRC Press.

Steinberg, A., Bowman, C., & White, F. (1999, April). Revisions to the JDL data fusion model. In *SPIE Conf. Sensor Fusion: Architectures, Algorithms and Applications III* (Vol. 3719, pp. 430–441). Retrieved December 9, 2017 from `http://bit.ly/2ylPOeU`

Toner, E. (2010). Creating situational awareness: A systems approach. *Institute of Medicine (US) Forum on Medical and Public Health Preparedness for Catastrophic Events*.

Tsarkov, D., & Horrocks, I. (2006, August). FaCT++ description logic reasoner: system description. *Proc. Third International Joint conference on Automated Reasoning*, 292–297.

Yalamanchi, A., Srinivasan, J., & Gawlick, D. (2003). Managing expressions as data in relational database systems. In *Proc. of the 2003 CIDR Conference* (p. 609).