

IPAS ontology development

David W. FOWLER, Quentin REUL and Derek SLEEMAN ¹

University of Aberdeen, Aberdeen, United Kingdom

Abstract. There is a trend in some manufacturing industries to move from selling products to providing services. As a result, designers must consider the life-cycle costs. In the aero industry, for example, this must be considered as well as weight, performance and manufacturing cost. The IPAS (Integrated Products and Services) project is intended to utilise Semantic Web technologies in order to provide feedback of information/knowledge acquired during operation of a product to the product's designers, and also to reuse knowledge from previous product designs. As part of IPAS, ontologies describing the products and processes have been created in order to allow service knowledge to be represented and shared. The design and implementation of these ontologies, and their planned future evolution, is described here. We also try to draw some lessons from our experiences.

Keywords. Ontologies, IPAS, design patterns

Introduction

In the past, manufacturers have designed products, manufactured them, and sold them to end users. There is a trend now away from simply providing products to including maintenance services. A good example of this is the Rolls-Royce TotalCare® package [1], where Rolls-Royce undertakes to provide flying hours to the customer at a fixed cost per hour. As the variable costs of operating the products is borne by the manufacturer (while the end user pays a fixed price), there is an added incentive to design the product in order to reduce overall lifecycle cost. To do this, the designers require to access knowledge from previous product service experience. This knowledge is contained in many different locations and types of documents. Additionally, we are concerned with making that information available on workstations attached to the company's intranet, and so we are exploring the use of Semantic Web technologies in the delivery of these services.

IPAS (Integrated Products and Services)² is a three year project co-funded by the UK Technology Strategy Board's Collaborative Research and Development programme³ and Rolls-Royce plc, that involves researchers from ten universities and companies, with specialisms in artificial intelligence, engineering, and organisational psychology. One of the principal aims of IPAS is to provide feedback from in-service operation of products to the design of new products and services. IPAS covers fields such as:

¹Corresponding Author: Computing Science, University of Aberdeen, Aberdeen AB24 3UE, United Kingdom; E-mail: d.sleeman@abdn.ac.uk.

²<http://www.3worlds.org/>

³<http://www.berr.gov.uk/dius/innovation/technologystrategyboard/index.html>

- Knowledge representation;
- Extraction of information from textual documents;
- Life cycle cost modelling;
- Process analysis;
- Social network analysis.

Within IPAS, we have mainly used ontologies to support the first two of these areas as part of Semantic Web-based demonstrators [2]. The intention of the project is that it will produce a specification for an integrated system to be fully implemented in the future.

The rest of the paper is organised as follows. Section 1 describes related work on ontology design, evolution and modularisation. Section 2 explains how the initial version of the IPAS ontologies was created. Section 3 describes the shortcomings of the ontologies, how they were redesigned in the light of this feedback, the principles that were used, and the ontology-based applications that have been developed in IPAS. In Section 4 we describe the remaining issues and further work, including plans for future evolution of the ontologies. Finally, in Section 5 we attempt to draw some conclusions.

Note on terminology. In this paper, to describe the ontologies developed during IPAS, the plural term *ontologies* has been used, whether they comprise one OWL file or several. These comprise several domain ontologies, covering engines, modules, materials, and so on. Each of these is referred to as *an ontology*. The only difficulty with using this terminology is that the initial version of the IPAS ontologies was held in one OWL file, and was previously referred to as *the IPAS ontology*. However, for consistency, and as it did cover more than one domain, in this paper we use the term *IPAS ontologies* for all of the versions. Furthermore, we use the term *part* to refer to all hardware (including engines and modules), and the term *component* for all parts smaller than a module.

1. Background

In this section we describe related work on how ontologies can be designed. Gruninger and Fox [3] propose an approach to engineering ontologies based on four steps. First the knowledge engineer defines the questions that the ontology must be able to answer. Secondly the vocabulary of the ontology is defined, i.e. its classes and properties. The third step is to use the vocabulary defined in the previous step to specify the definitions and constraints. Finally, the engineer tests the competency of the ontology by attempting to answer the questions defined in the first step.

Similarly, Noy and McGuinness [4] propose a methodology with the following steps (some terms have been updated to reflect OWL usage):

1. Determine the domain and scope of the ontology.
2. Consider reusing existing ontologies.
3. Enumerate important terms in the ontology.
4. Define the classes and the class hierarchy.
5. Define the properties of classes.
6. Define the restrictions on properties.
7. Create instances.

Noy and McGuinness stress that “ontology development is necessarily an iterative process”, and that as an ontology is a model of reality, it will have to be revised and debugged. This must happen when: (a) it is found that the ontology cannot answer an existing question or support an activity, (b) it is found that the ontology is inconsistent (some classes cannot have instances, due to logical contradictions), (c) the existing ontology appears to be correct, but could be restructured in a more logical fashion, (d) new competency questions are added which the ontology does not currently address, or (e) there are major changes to the company’s data model or business model.

Lopez et al. [5] propose a methodology, called Methontology, to provide a user-friendly approach to knowledge acquisition by non-knowledge engineers, and an effective method to domain-knowledge-model construction and validation without specifying a particular representational schema. The process comprises five steps, namely specification, conceptualisation, knowledge acquisition, integration and implementation. The Ontology Design Environment (ODE) performs the latter step.

2. Initial IPAS ontologies design

As the ontologies were required to extract data from text documents, and then to support working demonstrators (see Section 3.3 for details of these), it was decided to create an initial version of the ontologies quickly. This was then followed by a stage in which the ontologies were refined in the light of experience, but subsequently further versions of the ontologies were produced as the scope of the project expanded. Therefore, the initial version (v1.0) was planned for three months into the project, with a second version (v2.0) at nine months, v3.0 at fifteen months, and the final version, v4.0, two years after the project start. This meant that other partners in IPAS would be able to start developing their subsystems fairly quickly, with the understanding that early versions of the ontologies would be small, and subject to change.

It is important to realise that the scope of the ontologies was not really clear from the start, but was modified over time, as it became clearer what the software demonstrators were to do, and what activities they would support. The main activity supported by the demonstrators is to provide feedback from the operation of existing engines, and so we concentrated on representing existing engines. While for future systems it might be useful to represent future designs (consisting of abstract potential components, rather than existing ones), it has not been done in the current ontologies.

We chose OWL [6] as the language for ontology development because of the wide range of software tools that can be used with it, due to our previous experience with it, and also because it is an open, nonproprietary, standard (a W3C Recommendation). Also, as it was not clear from the start of the project exactly what the knowledge was to be represented, we could not decide whether OWL was fully adequate for the task, but we were encouraged by the fact that other (proprietary) knowledge representation efforts for engineering domains have also used OWL, e.g. PLCS [7] and ISO-15926 [8]. OWL has three sublanguages: OWL Lite, OWL DL and OWL Full, varying in expressiveness and tractability for reasoning (OWL Lite being the least expressive, but most tractable, and OWL Full being the most expressive, but least tractable). It was decided to use OWL DL as a tradeoff between expressiveness and tractability; this decision had important implications later. The main one is that often it is required to refer to a class rather than

to an individual (i.e. as the value of a property). OWL Full allows reference to classes, whereas OWL DL does not.

The initial versions of the ontologies were intended to cover the event reports (describing incidents that led to maintenance actions being taken) that were to be fed back to the designers, and also the strategy sheets for components (plans for mitigating deterioration mechanisms for components). To ensure that the ontologies could represent the principal concepts in these documents, some sample reports were examined, and the key concepts extracted manually. Our intended approach for developing ontologies largely followed the approach of Noy and McGuinness [4] (although this predates the introduction of OWL), supplemented with insights from other sources such as [9] which is specifically about OWL (and recommends an approach that allows for modularisation, although doesn't give details as to how this can be done). It is fair to say that we did not fully appreciate at the start of the project the importance of modularity in ontologies, and the early versions of the ontologies were monolithic, with everything stored in one OWL file. This caused problems that had to be addressed later (see Section 3.1).

It was clear that it would be necessary to represent engines and their components. There are several distinct types of entity that could be represented, including: (a) actual physical parts, i.e. those that have been manufactured and exist as distinct individuals, (b) specific, existing, *types* of part (used in cases where it is necessary to refer to a type of part, but not a specific individual part, and (c) a more generic type of part, not necessarily existing (perhaps a part that is being designed). For the ontologies so far, we have concentrated on the first two of these types (namely, existing parts). Version 1.0 of the ontologies included a very small, hand-built, ontology covering a few parts only, but for more extensive uses (including the subsequent demonstrators), many more part types would be required. To build this ontology from scratch would have been prohibitively expensive (the current version of the ontologies has around 3000 types of part, covering nine engine types). Rolls-Royce provided a taxonomy in the form of a series of linked HTML pages that had been developed for a previous project, and contained a model of an engine.

This taxonomy covered a wide range (e.g. parts, processes, materials, etc.), but there were several problems with it. Firstly, different types of relationship were represented with the same taxonomic relation. For example, in the engine section, the links were part-of; in the materials section the links were is-a; in the process section they were sub-processes (or temporal part-of). Secondly, the engine section of the taxonomy was generic, covering a typical engine rather than any specific engine(s). This may be sufficient for some purposes, but not for applications where comparisons are made between engine types. Finally, there is no further documentation associated with any of the nodes in the taxonomy. Therefore, the only explicit information is the names of the nodes, and the links between them (and the exact nature of these links has to be inferred). However, as it was the best information source available at the time, and machine-readable, the section of the taxonomy dealing with the engine was used as part of the early ontologies, with the taxonomic links being replaced by part-of relations. It should be emphasised that we were not representing a complete engine design with a complete breakdown of parts, as this would require a means of representing multiple instances of the same part being part of an assembly. Instead, we used an "is-allowed-to-be-part-of" relationship only.

Another major section of the ontologies covered deterioration mechanisms. These are the processes by which components deteriorate, and eventually are either replaced

after failing an inspection, or fail to perform their task. Deterioration mechanisms are clearly important in IPAS, as the causes of failures or replacements are to be recorded, and fed back to designers. The initial deterioration mechanism ontology was created by referring to a *Mechanism Prompt List* used at Rolls-Royce. The Mechanism Prompt List was intended to remind designers of various possible mechanisms while considering a new design, and comprises main categories (e.g. Fatigue, Creep, Adhesive wear) and subcategories (e.g. for Fatigue: Fretting, Corrosive, Rolling Contact, etc.). The Mechanism Prompt List was used to construct the deterioration mechanism ontology fairly straightforwardly. The main categories were used as concepts in the first level of the class hierarchy. The subcategories were used as subclasses of the main classes. Some minor clarifications were carried out by referring to engineering textbooks, and by discussing the class hierarchy with design engineers.

3. Re-engineering the ontologies

After the initial ontology development had been completed, time was allocated to (a) get feedback from other partners, (b) to reflect on the design of the ontologies, and (c) to gain access to more documentation to help in enhancing the ontologies. Feedback from the other partners was difficult to get, probably due to a steep learning curve in using Protégé and OWL for many. Automatically produced HTML documentation for the ontologies (OWLDoc [10]) and graphical representations of the class hierarchy of the ontologies were used to try to overcome this problem, but with limited success. Most feedback came later in the project, when the demonstrators were implemented (see Section 3.3 for details). However, the ontologies were enhanced by reviewing their design, and attempting to restructure them by using design patterns [11,12] and by reflecting on best practices. Design patterns allow the use of good solutions to common ontology design problems to be reused (the solution may be more efficient, conceptually elegant, or allow reasoning to be performed). Also, gaining access to more documents provided resources to allow the ontologies to be enlarged.

The conclusions drawn immediately were firstly that the ontologies would have to be split into modules for them to be manageable as they grew, and secondly that ontology design issues would have to be examined carefully. Examples of such design issues were most obvious in representing engines and components (discussed below). The initial ontologies were not capable of such representations, and had to be redesigned. Following the appraisal of the initial version of the ontologies, further versions were produced, using better design practices and the additional sources of knowledge that had become available.

3.1. Ontology modularisation and evolution

The initial version of the ontologies was monolithic, in that everything was contained in a single OWL file. This was not a major problem while the ontologies were small, but when the engine ontology was added, it became clear that the single ontology file would have to be separated into several smaller domain ontologies. This process was repeated during the evolution of the ontologies, with the engine ontology being divided into separate engines, modules and parts ontologies. Also, ontologies describing different

Table 1. The evolution of the IPAS ontologies

	v1.0	v2.0	v3.0	v4.2
Classes	72	452	375	7863
Object properties	30	36	15	78
Datatype properties	34	60	23	124
Files	1	2	3	14

types of reports were subdivided. This modularisation had the advantages that individual ontologies could be worked on independently and reused.

Table 1 gives an overview of how the ontologies have changed since the beginning of the project. The drop in size between v2.0 and v3.0 is due to the ontologies being radically overhauled, due to (a) a redesign using ontology design patterns, and (b) a better understanding of what the ontology should describe (the “early prototype” approach to building ontologies in this project had the advantage of producing early versions for inspection and use, but the disadvantage that the scope of the ontologies was not fully understood while the ontologies were being built) . There was a large increase in size between v3.2 and v3.3 as new sources of component data became available and automated methods were developed to extract this data.

One of the ontology design patterns that was most frequently used in building the IPAS ontologies is the use of classes as property values. This construct is used (a) to refer to a generic (or typical) engine or part rather than a specific physical object, (b) to refer to deterioration mechanisms or materials, which logically seem to be better represented by classes than individuals, and (c) to refer to specific parts that are not “trackable” (they do not have a serial number, so it is not possible to distinguish subsequently between two individuals of the same class . In [13], several possible approaches to work around the OWL DL restriction of not being able to use classes directly as property values are given. The second of these approaches was chosen, which involves creating a “dummy” individual for each class that may need to be referred to. The main reason for the choice was simplicity; this may be reviewed at a later stage. To achieve the effect of using a class as a property value, all that needs to be done is to refer to the dummy individual instead. This also has the additional benefit of allowing assertions about classes. In OWL, classes can be in the domains of special kinds of properties, called *annotation properties*, but these cannot have restrictions placed upon them (so, for example, the type or allowed values of an annotation property’s range cannot be specified). By using a dummy individual to stand for the class, the properties of the class can be represented as the properties of the individual.

Representing the materials that parts are made of provides an example of how the approach was used in the IPAS ontologies. In an earlier version of the parts ontology, the material that a generic part is made of was indicated by an OWL annotation property (*has_material_code*). The class for the part type has this property, and its range is a string (the three letter material code). See Figure 1 for a representation of this approach. This was clearly not ideal, but this is forced upon us as classes can only have annotation properties, which cannot have range restrictions. To find the material in the materials ontology, a search would have to be performed on the material code. The current design revised the parts ontology so that each part class has a dummy individual associated with it, and uses a *has_material* property to link this class to an individual from the

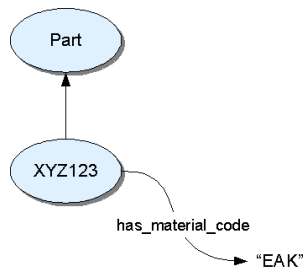


Figure 1. An early ontology design to represent the material of a component using annotation properties

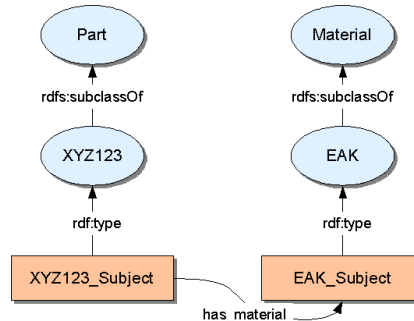


Figure 2. A more sophisticated design using classes as property values

materials ontology (Figure 2). In this, and in other ontologies, the dummy individuals were added by processing the OWL-DL ontology with a simple Java program.

As an example of modularisation, semi-automated ontology construction, and using classes as property values, we consider the development of the ontology describing engines, modules, and their components. The initial ontology was divided into three sections:

Engines the top level entities;

Modules one of a small number of interchangeable parts in an engine;

Components anything below the level of a module, including indivisible parts, and assemblies.

The engine is the highest level entity that we are concerned with in the parts ontologies. We have currently dealt with versions of the RB211 Trent engine.

The Trent engine has eight main modules. The modules ontology has one class and one dummy individual for each of these module types — we are currently not aiming to represent every subvariation of module that exists.

The components ontology was built up using parts lists supplied in comma separated value (CSV) format. Each row contained an entry for a particular part (including parts that were versions of older parts). The details for each part type includes:

ATA location This is a multi-field entry that identifies the particular part on a drawing;

Part number The company’s code number for all parts of this type;

Category A high level description, such as “VANE”, “BOLT”, etc. This field was limited to eight characters;

Subcategory A further description — “BIHEX HD” is an example of a subcategory for the category “BOLT”. Around 50% of the entries had a subcategory entry, the rest did not;

Detail description A final text description, often giving dimensions, such as “.190 DIA X .438” for a “BIHEX HD BOLT”. Around 40% of entries had an entry for this field;

Material code A three letter code used by the company to identify materials;

Service bulletin number An ID number of a document relating to the part, often describing modifications that were required to update the part from a previous version.

The category and subcategory fields were used to create a shallow class hierarchy for the parts. The part number was used to make a new leaf class in the hierarchy. Dummy individuals for the leaf classes were created, and the other fields in the spreadsheet were used to fill in the property values for these.

As part of the components ontology, we also represented *features*. Features are the locations on a part, usually with some functional purpose. They are important for our purposes, as deterioration or damage is often described as occurring on a particular feature of a part. We do not currently represent features in detail, instead just the name of the feature is used. For example, we record that a HP NGV has a feature called “trailing edge”, but not its size, or relation to other features. We used SKOS [14] to represent the names of features, and their synonyms. Features will be put into a separate module in future versions of the IPAS ontologies.

3.2. Other IPAS ontologies

A materials ontology is used to represent the materials that components are made of, together with some of their physical and economic properties. The ontology imports two other existing ontologies to describe physical units⁴ (for the temperature, density, and tensile strength properties), and currencies⁵ (for the cost property). The physical units ontology allows for units in different systems (e.g. metric and British Imperial) to be related. For example, `gram` and `pound` are both individuals of `MassUnit`, and properties `hasUnit` (objecttype) and `factor` (datatype) are used to relate units. So, to assert that a pound is equal to 453.59238 grams, the following statements are made:

```
pound hasUnit gram
pound factor 453.59238
```

The deterioration mechanism ontology remained broadly similar to the original version (although it is now a separate module in the IPAS ontologies).

A new ontology describes *mechanism records* — a means of storing information from service records about deterioration mechanisms and their actual occurrences, and/or details of how the mechanism was “designed against” (i.e. how the designers prevented or reduced the occurrence of the mechanism; this is clearly important knowledge to store, as otherwise future designers may unwittingly reintroduce the mechanism). The term *mechanism template* is used to describe the format of the mechanism records — that is, a mechanism template is a class, and the mechanism records are instances of that class. The mechanism template ontology was developed by having engineers describe the types of knowledge that they thought would be required to help designers design future components. For the initial version of the ontology, many of the more complex fields of the mechanism template were left as free text entries (for example, the details of mechanism reduction/prevention is represented as a collection of strategies for inspection, maintainability, repairability, etc., but each of these strategies is free text). In later versions, these may be replaced by more structured alternatives.

⁴<http://archive.astro.umd.edu/ont/index.html> from the Department of Astronomy, University of Maryland

⁵<http://www.daml.ecs.soton.ac.uk/ont/currency.owl> from the University of Southampton

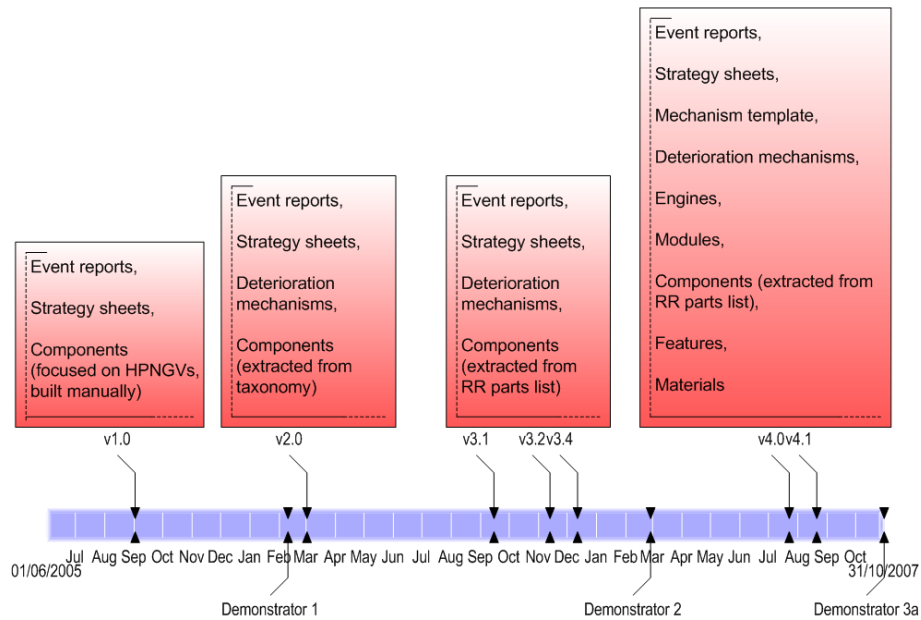


Figure 3. A timeline of the ontology development and the IPAS demonstrators

3.3. Applications of the ontologies

The IPAS project's deliverables include several demonstrators which are intended to illustrate the use of semantic web technology to retrieve service data for use in design. Currently two such demonstrators have been completed; work is in progress on a third. The first demonstrator allowed users to plot graphs from service data, (such as numbers of service events per engine type).

The second demonstrator supported more complex searches for individual service events, and retrieved service events for different deterioration mechanisms. This demonstrator also retrieved images of damaged parts. This demonstrator uses data that was extracted in the form of RDF triples from service event reports, and allows the user to search for reports based on the engine and part types.

The third demonstrator is intended to handle mechanism records, which will enable domain users to find details of previous events, to create new mechanism records, and search for existing ones. The user (a senior engineer, or "knowledge guru") will take a more active role in creating knowledge that other users (e.g. designers) will then be able to search through.

Figure 3 shows a timeline of the IPAS ontology development, together with the demonstrators that were implemented at the time of writing (the third demonstrator is being implemented in three smaller stages: 3a, 3b and 3c).

4. Remaining issues and future work

In this section we describe some outstanding issues concerning the design of the ontologies, and possible approaches to solving them. There are several problems with the

current IPAS ontologies, including: the cleaning of the ontologies that have been built semi-automatically; whether to relate the ontologies to an upper ontology; and how to represent change over time.

A significant disadvantage of the automated method of creating the parts ontology from a spreadsheet is that the spreadsheet contains names which lack consistency, (e.g. naming and terminating conventions have not been followed consistently), and these inconsistencies need to be resolved in some part of the process. For example, the pairs of terms `ADAPTER` and `ADAPTOR`, and `ARRGT` and `ARRNGMNT`, are found in the parts spreadsheet, and presumably refer to the same entities. It would be possible to use string comparison algorithms to suggest likely occurrences of this problem, but the final decision would have to be made by a domain expert. Ideally, checks would be implemented to ensure that these variants cannot occur in the first place. We are investigating an approach using `CleOn` (formerly called `CleanOnto`) and `SKOS` to do this [15].

Currently, there is not much inference being done with the IPAS ontologies — the implemented demonstrators use the ontologies to connect separate data sources. The main inferencing is simple transitive reasoning, whereby if an individual is a member of a class, it can be inferred that it is also a member of all superclasses of that class (this reasoning is handled by the Sesame triplestore). The example of parts having multiple names would be one simple example where inferencing could be used, by stating that two or more classes are equivalent. This is being done in the latest version of the components ontology. Another example would be in classifying events according to severity or importance. We also have to investigate the scalability of any reasoning, as the ontologies are rather large. One approach would be to use approximate reasoning, as in Pan and Thomas [16].

The choice of design patterns may be revisited — the choice of dummy individuals to represent classes has the advantage of simplicity, but may have disadvantages if more reasoning is to be performed.

The use of upper ontologies should be considered. An upper ontology is a high level ontology, not specific to any single domain. It would be possible to use an existing upper ontology that describes high level terms such as `Physical thing`, `Process`, `Person`, etc., that could be used as superconcepts for the IPAS ontologies terms. The advantage of this approach is that other existing domain ontologies that refer to the same upper ontology could be easily integrated with the IPAS ontologies. However, there are several competing upper ontologies available, and there is no commonly agreed standard (although they do have broad agreement on the most fundamental categories). For the moment, the IPAS ontologies remain as domain ontologies with no upper level ontology. For any future development however, it will be important to make use of an upper ontology (as one reviewer put it, this is key to avoiding certain well known ontological mistakes).

The representation of change over time has not been fully resolved. There seem to be two main approaches for this issue — three-dimensionalism (3D) and four-dimensionalism (4D) [17]. The 3D approach regards objects as existing only at the present time, whereas the 4D approach regards objects as having a presence that extends through space-time. One 4D upper ontology that has been developed is that of ISO Standard 15926: “Lifecycle Integration of Process Plant Data Including Oil and Gas Production Facilities” [18], which has also been translated into OWL. However, Smith [19] has pointed out its many defects, although these do not necessarily invalidate the 4D ap-

proach. For pragmatic reasons (the IPAS ontologies were developed without the knowledge of these 4D ideas, and there is insufficient time to make further significant revisions before the end of the project), a 3D approach has been taken. So, the composition of generic engines can be represented, as can the current composition of a specific engine. To record which specific parts are in which specific engines, some new classes and individuals will be required — either to represent the state of an engine at a particular time, or (as is currently done) a series of reports of removals and replacements, from which the state at any time can be reconstructed.

5. Conclusions

Here we have tried to collect together some of the lessons that were gained during the process of developing the IPAS ontologies, and in developing the subsequent demonstrators.

Firstly, previous experience and best practice in designing ontologies should be drawn upon, as a bad choice in ontology design can make future updating very difficult. Modularity of ontologies should be planned for from the start. For example, the original Parts ontology was badly designed, as one large ontology rather than separate engines, modules, and components ontologies. Also, the original design choices made it possible to represent a particular engine type, but very difficult to represent additional engines.

Secondly, the “early prototype” approach to building ontologies in this project had the advantage of producing early versions for inspection and use, but also had the disadvantage that the scope of the ontologies was not fully understood while the ontologies were being built.

Thirdly, difficulties were encountered in persuading other partners in the value of ontologies. This may be due to several factors: (i) unclear explanations of ontologies on our part; (ii) the long lead time for developing substantial and stable ontologies contrasting with the quick results expected by others; (iii) confusion as to what the ontologies contained and what functions they could support, due to difficulties in installing and using software such as Protégé.

Finally, the web service implementation of the completed IPAS demonstrators went fairly smoothly, with web services being provided by three of the university partners, and integrated at a one day meeting. This seems to indicate that although the Semantic Web infrastructure may be complicated to set up, it is fairly straightforward to extend.

Acknowledgements

This work is co-funded by the UK Technology Strategy Board’s Collaborative Research and Development programme⁶. Our thanks go to the other IPAS partners for their feedback. We also thank the anonymous reviewers for their helpful comments.

⁶<http://www.berr.gov.uk/dius/innovation/technologystategyboard/index.html>

References

- [1] Rolls-Royce plc. Rolls-Royce TotalCare services webpage. Available at: <http://www.rolls-royce.com/service/civil/totalcare/default.jsp>.
- [2] S. C. Wong, R. M. Crowder, G. B. Wills, and N. R. Shadbolt. Knowledge engineering - from front-line support to preliminary design. In D. F. Brailsford, editor, *Proceedings of ACM Symposium on Document Engineering (DocEng)*, pages 44–52, Amsterdam, The Netherlands., 2006.
- [3] M. Gruninger and M. S. Fox. Methodology for the design and evaluation of ontologies. In *Proceedings of the Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI, 1995*.
- [4] N. F. Noy and D. McGuinness. Ontology development 101: A guide to creating your first ontology. Technical Report KSL-01-05, Knowledge Systems Laboratory, Stanford University, 2000.
- [5] M. F. Lopez, A. Gomez-Perez, J. P. Sierra, and A. P. Sierra. Building a chemical ontology using Methontology and the OntologyDesign Environment. *IEEE Intelligent Systems and Their Applications*, 14(1):37–46, 1999.
- [6] W3C. OWL Web Ontology Language Guide. Available at: <http://www.w3.org/TR/owl-guide/>, February 2004.
- [7] D. Price and R. Bodington. Applying Semantic Web Technology to the Life Cycle Support of Complex Engineering Assets. In *Proceedings of the ISWC-2004*, pages 812–22. Springer, 2004.
- [8] R. Batres, M. West, D. Leal, D. Price, and Y. Naka. An Upper Ontology based on ISO 15926. *Computers & Chemical Engineering*, 31(5–6):519–534, May 2007.
- [9] A. Rector, N. Drummond, M. Horridge, J. Rogers, H. Knublauch, R. Stevens, H. Wang, and C. Wroe. OWL Pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In E. Motta, N.R. Shadbolt, A. Stutt, and N. Gibbins, editors, *Engineering Knowledge in the Age of the SemanticWeb: 14th International Conference, EKAW 2004*, pages 63–81. Springer, October 2004.
- [10] CO-ODE. OWLDoc Home Page. Available at: <http://www.co-ode.org/downloads/owldoc/>.
- [11] A. Gangemi. Ontology design patterns for semantic web content. In *Proceedings of the Fourth International Semantic Web Conference (ISWC-05)*, 2005.
- [12] M. Egana, R. Stevens, and E. Antezana. Ontology design patterns for bio-ontologies. In *Bio-Ontologies Meeting, ISMB 2007*, Vienna, Austria, July 2007.
- [13] W3C. Representing classes as property values on the semantic web. Available at: <http://www.w3.org/TR/swbp-classes-as-values/>, April 2005.
- [14] W3C. Simple Knowledge Organisation Systems (SKOS) - Home Page. Available at: <http://www.w3.org/2004/02/skos/>.
- [15] Q. Reul, D. H. Sleeman, and D. W. Fowler. CleOn in IPAS: Resolution of lexically incoherent ontologies. Technical Report (forthcoming), University of Aberdeen, 2008.
- [16] J. Z. Pan and E. Thomas. Approximating OWL-DL Ontologies. In *the Proc. of the 22nd National Conference on Artificial Intelligence (AAAI-07)*, pages 1434–1439, 2007.
- [17] T. Sider. *Four-dimensionalism*. Oxford University Press, 2001.
- [18] M. West, C. Partridge, and M. Lycett. Enterprise data modelling: Developing an ontology-based framework for the shell downstream business. In *Formal Ontology Meets Industry (FOMI 2006)*, 2006.
- [19] B. Smith. Against idiosyncrasy in ontology development. In *Proceedings of the International Conference on Formal Ontology in Information Systems (FOIS 2006)*, Baltimore, Maryland (USA), November 2006.