Beyond Ontology Construction; Ontology Services as Online Knowledge Sharing Communities

Yang Li, Simon Thompson, Zhu Tan, Nick Giles, and Hamid Gharib

Intelligent Systems Lab, BTexact
MLB1/PP12, Orion Building
Adastral Park, Ipswich, UK,IP5 3RE
{yang.li,simon.2.thompson,zhu.tan,nick.giles,hamid.gharib}
@bt.com

Abstract. There are a number of different ontology server implementations available, Their functionality focuses on editing, browsing and storing ontologies. In some cases the ontology server also provides an inference engine that allows statements about the relationships between entities in different ontologies to be tested or retrieved. These functions are certainly required for a practical deployment of an ontology service in an organization but when an ontology service is to be deployed in an open environment, such as the Semantic Web, a number of other considerations become apparent. This paper describes ACOS (Agent Cities Ontology Service), during the development of ACOS we have explored some aspects of creating a collaborative, community-oriented ontology server, in an open environment.

1 Introduction

Ontology servers create a collaborative environment for managing, developing and sharing ontologies. Without such an environment, it is much more difficult for agents to interconnect their ontologies. Ontology servers enable the exchange and re-use of ontologies with minimal effort, possibly facilitating their use within practical system development environment [1].

In this paper we have developed a technically novel ontology service that introduces some new reasoning facilities to verify the consistency of ontologies and to support their construction and the construction of relationships within the ontologies (intra ontology relationships) and between ontologies (inter-ontology relationships). Simultaneously we have developed a management system for ontologies that draws inspiration from the Slashdot web site's method of managing comments on stories [9].

Within this environment, users are willing to use and develop ontologies collaboratively. In turn, ontologies become well developed and attract more users. The ontology server allows users to control access to their ontologies, such as allowing only specified users to read their ontology, or allowing some users to alter their ontology. In addition, more flexible access controls can be used, allowing certain categories or qualities of users a set of permissions. For instance, one can allow highly rated users to modify ontologies, on the basis that they are acknowledged to be good at managing other ontologies, and can therefore assist in the development of

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 469–483, 2003.

yours'. User ratings might be based upon the level of use of the ontologies that they own, allowing those users that produce ontologies the community needs to be recognized.

Ontology management is clearly important, but it is not clear who should do it, or how they will be paid. In the WWW where information repositories are used by millions, their management is paid for only by advertising, or in some cases by state compelled subscription (CCTV, BBC). Trusted knowledge sources are significant and useful, but no one is willing to pay for them.

This is not to say that there is no money to be made from providing an ontology service. It is our belief that a subscription to access the shared space would be acceptable to its users, provided that a premium is being obtained by the subscriber. We think this because the ontologies are business critical resources. While there is no study to confirm this, our intuition and experience of open system development for example during the construction of the demonstration applications during the Agentcities project, shows that an ontology is not only central to the local development effort, it is central to ensuring that the developed system is interoperable with other systems. This value means that there is a business case for paying for access to an ontology server.

The question is: how much? The cost of a resource is determined by the cost of the infrastructure required to provide it and the cost of the effort that will maintain it. Those that deploy an ontology service will rapidly discover (as we have) that there is a great deal of work to be done in deleting test deployments, checking updates are acceptable, retrieving lost data and so on. The work is comparable (unsurprisingly) to the work that a DBA undertakes. It is therefore significant to note that there are no public sql databases, accessible for query and update, that are in widespread use for development in internet systems. Our point is: in terms of private resources for in house development efforts a business case for an highly managed ontology server can be made. As a mode of service provision to a community that is paying via subscription it is much harder to see that this is the case.

It is this insight that motivated us to develop an automated consistency checking and community based management algorithms that are used to maintain the ACOS ontology server. These mechanisms mean that the users who need the ontologies: those that make most use of them and have the highest stake in them are able to maintain them.

The contributions that we report here are:

- An ontology service that uses a specific set of semantics for the construction of ontologies and inter-ontology connections.
- An ontology service with novel validation & consistency checking abilities.
- An ontology service with a novel management interface.

The rest of this paper is organized as follows.

- Section 2 introduces the design and infrastructure of our ontology server; it discusses the knowledge representation that we chose and the mechanism for connecting ontologies.
- Section 3 describes the mechanism which is used to construct individual ontologies.
- Section 4 describes the consistency checking capabilities of ACOS

- Section 5 describes the mechanisms for community based management of the ontology service
- Section 6 is a comparison between ACOS and other ontology servers
- In section 7 we draw conclusions from the work that we have done and suggest directions for future work.

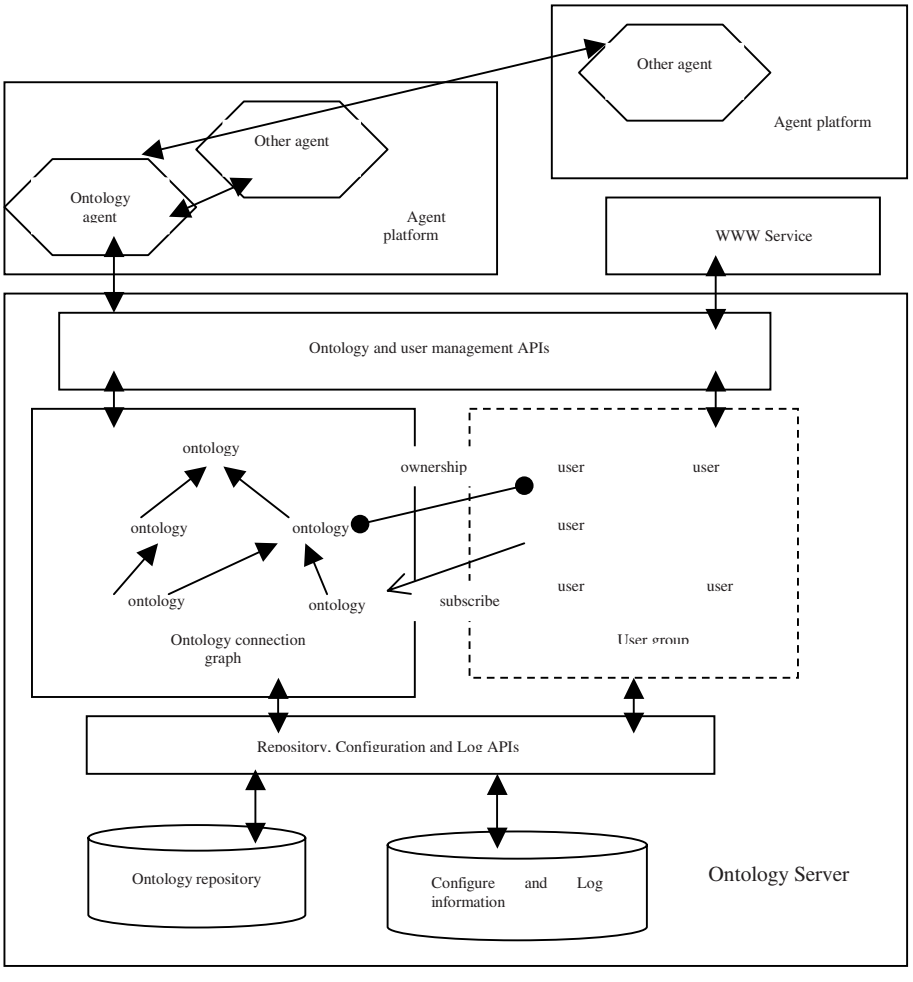

Fig. 1. ACOS Architecture

2 ACOS Architecture

The architecture of the ACOS sever is shown in Figure 1. Two central data structure held in the ontology server are ontology connection graph and user group. Ontology connection graph is the internal ontology representation managed by the ontology server; whilst the latter is the data structure used for user access control management.

These two components are linked with each other through a system of ownership and subscription relations. Ontology and user access management APIs are the implementation of interfaces that provide functions of ontology management and user access control management to external clients such as a web browser user or a software agent. In the latter case, an Ontology Agent will act as a middle-man, connecting between ontology server and other agents, using agreed Agent communication language. The other agents can reside on either the same or different agent platforms. Repository, configuration and log APIs provide functions to put the in-memory data into persistent storage, in our case, files in the format of DAML+OIL and XML in the hard disk. In the rest of this section, we will discuss the internal ontology representation.

2.1 Knowledge Representation

Designing a suitable knowledge representation is the first step towards building an ontology server. Various knowledge representation schemas have been proposed including logic, semantic nets, neural nets, production rules, cases, frames & object-oriented representations [2]. In 1970s and 1980s, the booming of expert systems highlighted production rules & frames, and made them popular in both knowledge engineering and software engineering communities [3]. This trend has bled into the semantic web community, as AI and Networking communities are collaborating to bring knowledge representation and reasoning capabilities to the WWW [4, 5, 6].

An expert system is designed for capturing special domain knowledge and using it, in combination with an inference engine, for simulating the automatic reasoning process of a human expert. The knowledge embedded in the expert system is not used for knowledge sharing across diverse communities. Some researchers have raised doubts that this change in the context of use of the knowledge system, from single interactions in a closed community of knowledge to interactions by many users in an open environment may render knowledge representation, such as Frames [7, 5], inappropriate for use in the semantic web [8, 10]. A typical example from the W3C for informally describing "person", using DAML+OIL, is: "*a person is a subclass of animal, whose parent is also a person, who has only one father, and whose shoe size is at least 1"* [11]. The question is: if another agent defines a person as "*subclass of animal, whose parent is also a person, who has only one mother, and whose hat size is at least 1",* are these concepts of *Person* compatible? Can an interaction between these two agents be conducted?

The Foundation for Physical Agents (FIPA) is a standards body which has developed interaction standards for agents in open environment. During the production of its standards set FIPA identified ontology integration problem as a major issue for communicating agents [6]. It believes that "even if two agents adopt the same vocabulary, there is no guarantee that they can agree on a certain information unless they commit to the same conceptualization" [6]. Assuming that each agent has its own conceptualization, "a necessary condition in order to make an agreement possible is that the intended models of both conceptualization overlap" [12].

Accordingly, FIPA introduced an important classification dimension for ontologies – their level of detail: a *fine-grained ontology* that has very rich axioms, written in a very expressive language like full first order logic, gets closer to specifying the

intended meaning of a vocabulary, but usually very hard to develop and hard to reason on; a coarse ontology that consists of minimal set of axioms written in a language of minimal expressivity, to support only a limited set of specific services [6]. These two categories are also termed as *off-line/reference ontologies* and *on-line/shareable ontologies*, respectively: the former are only accessed from time to time for reference purposes, while the latter support core system's functionality that is the overlap of local ontologies [13].

ACOS is designed to be an on-line community in which a diverse of software agent groups can contribute and use ontologies at run time. A key enabler to this scenario is a high degree of "shareness" of the ontologies maintained by the server. To achieve shareness, it is required that local features of an ontology be removed. Example constructors that can bring in local features include *property* in DAML+OIL [5] or *slot* in Ontolingua [7] that can be used to describe product attributes [8]. It is normally easy to define and agree on a particular product that can be represented as a class in a Framework language; it is, however, hard to agree on the attributes of a product. For instance, a mobile phone model produced by Motorola may contain features that Nokia does not provide or vice verse; can we still recognize a model as a Mobile Phone, if some of the product features are missing in that model? In this case, we tend to treat products as shareable ontologies and attributes reference ontologies.

Other example constructors that bring in local effect are *part-of* in Framework [7] and *disjoint-with* in Description Logic [14]. Constructor part-of is normally used to specify the compositional feature of a product/process. Such a composition is often due to a local requirement, for instance, a network card needs to be installed on a computer or using a swipe card is needed to get access to a building. A computer without a network card or getting access to a building without using a swipe card does not deny the fact that it is still a computer or a process of getting access to a building. Constructor disjoint-with is normally used to specify the property that two classes are disjoint with each other. An example can be found in W3C: (disjoint-with Man Woman) [11], which indicates that Man should be disjoint with Woman. As more discoveries are made in science, and as society and culture gradually advance, the boundary between Man and Woman could become blur, i.e., the above property may not hold given changed definitions on Man and Woman.

To achieve a high degree of shareness, we need to capture ontologies that are in coarse-grain and are more stable in the process of evolution. We believe that constructors *Class*, *subClassOf* offer such good features. Constructor Class is used to specify a concept in a coarse-grain (compared to attributes that are in fine-grain); Constructor subClassOf is used to document the evolutionary history of concepts and history, having time as its horizon axis, is resistant to changing its past. In addition to Class and SubClassOf, we also introduced *SuperClassOf*, *sameClassAs* and *instanceOf* as the constructors in ACOS. The definitions of these constructors are in accordance with those described in FIPA ontology service specification [6]. We will not present them separately.

2.2 Constructing an Ontology Hierarchy

To facilitate collaborative, on-line ontology development, we implemented *import* mechanism that is equivalent to *extension* relationship described in the FIPA ontology service specification [6].

In a collaborative development, an ontology developer normally focuses on a particular sub-domain. For instance, a restaurant ontology developer may be only interested in building concept hierarchy related to restaurant industry rather than automobile industry. In the meantime, to build sound restaurant ontology, substantial supporting knowledge is needed from food industry, cookery industry, interior design engineering, management science, etc. A component architecture can best suit for this scenario, where individual domain expert develops ontology in a particular sub-domain and reuse the ontology developed by other domain experts.

Informally, the semantics of "Ontology O1 imports O2" is that (1) all the concepts and instances that are defined in O2 are also found in O1; (2) all the relationships between entities in O2 are also found in O1. It is noted that the constructor import is transitive, i.e., if "O1 imports O2" and "O2 imports O3", we also have "O1 imports O3" or "O1 imports both O2 and O3". It is also noted that circular dependencies are allowed in the import chain. For instance, while restaurant ontology needs support from management science ontology for describing its human resource, the management science ontology may also needs support from restaurant ontology for describing health and safety issues. Formally speaking, the import relationship constructs a *cyclic graph*. To identify all the supporting/imported ontologies of O1, we can simply compute the transitive closure of O1 by following the import chain.

Seemingly an easy mechanism, "import" has intrinsic implication in maintaining the consistency of ontology system, even in the ACOS where ontology constructors are simplified, to a maximum degree. By gluing disparate ontologies together, one faces the issue that properties in different ontologies may contradict with each other. The issue becomes even serious when changes are made in an existing ontology that may invalidate many properties of the ontology system that previously hold.

The issue of knowledge base maintenance was well documented in the field of knowledge engineering. Three strategies for consistency-checking are notable: (1) allowing the existence of "inconsistency" by relaxing the criteria of consistency-checking; (2) reducing the effort of consistency-checking by enforcing modified model that can control the ripple effect [15, 16]; (3) resolving inconsistency by allowing humans intervene, in workshops [7].

To facilitate collaborative, on-line ontology development, we employed a fourth strategy: automatically checking the consistency of ontology system in the process of ontology development. The algorithm for consistency checking will be discussed in Section 4.0. In addition to facilitating ontology construction, the import mechanism is also used to evaluate the relative importance of ontologies. In the next section we present the ontology construction services that are provided to ACOS users.

3 Ontology Development Services

In conformance to FIPA ontology service specification [6], ACOS also provides functions that allow software agents to assert, retract and query on predicates. The predicates are in the form of (subClassOf C1 C2), (superClassOf C1, C2), (sameClassAs C1, C2) and (instanceOf I1, C1) that were described in Section 2.

The semantics of action ASSERT, as specified by FIPA, is to "add, create, or define the said predicate in an ontology. The syntax of ASSERT action is as follows: (ASSERT (predicate)). The ontology agent is responsible to respect the consistency of

the ontology and it can refuse to do the action if the result would produce an inconsistent ontology" [6]. In addition to preserving this semantics, we also make two extensions of ASSERT in the Agentcities ontology service, which are tailored for needs of facilitating collaborative, on-line agent community. First, allowing asserting predicate that is involved in two ontologies. Second, refining the result "failure" for action ASSERT into two reasons: "the predicate already explicitly exists", "the predicate is already implied". The former capability facilitates component-based ontology development; the latter is used to evaluate and manage agent's behaviour. For instance, a user who asserts a predicate that has already explicitly existed in an ontology may be less experienced than the user who asserts a predicate that can only be implied by the ontology. An analogy to this scenario is the difference between copyright and patent, where identifying infringement of a patent is undoubtedly more difficult than claiming the violation of a copyright.

The semantics of action RETRACT, as specified by FIPA, is to "remove, delete or detach the said predicate in the ontology definition. The ontology agent is responsible to respect consistency of the ontology and it can refuse to do the action if the result would produce an inconsistent ontology" [6]. Similarly, we also extend the semantics of RETRACT by allowing a predicate that is involved in two ontologies be retracted and refining the result "failure" for action RETRACT into two reasons: "the predicate does not exist", "implied predicate is not retractable".

Once an ontology system is developed, queries can then be made to the ontology server asking about the relationship between classes or between a class and an instance, the ontologies that contain a certain class/instance, the subclasses/superclasses of a given class, etc. The applications of these ontology services are significant in the real world. For instance, a shopper at a supermarket who intends to buy a pack of apples may end up with having a bag of oranges in his/her hand! The real world is full of fuzziness & uncertainty and well-organized ontology systems should help to accommodate it.

In addition to the functions of ontology construction and query, the ACOS also offers services of uploading and downloading ontologies. Currently we support DAML+OIL as an external format.

4 Consistency Checking Capability

As mentioned earlier, consistency-check is essential to ensuring the integrity of an ontology system. Informally, we say an ontology system is consistent iff there is no contradiction between classes/relationships in the ontology. The consistency-check mechanism of ACOS is built on top of its query capability. A key query function is to decide whether the predicate (subClassOf C1 C2) holds. Agrawal developed an efficient algorithm for deciding the "is-a" relationship between entities by computing a "compressed transitive closure of a graph". To compute the compressed transitive closure, a post-order traverse of the graph is needed, which incurs a computational complexity of O(n), where n is the size of the graph. Once the closure is in hand, only a simple integer range comparison is needed to decide whether the is-a relationship holds or not. This result generally applies to acyclic graph and the compressed transitive closure is incrementally maintainable, at a higher cost [17].

There is always a trade-off between the effort spent in using a structure and that in maintaining the structure. Another common method for deciding the is-a/subsumption/subClassOf relation is to trace through the relation chain. In this case, the effort of maintaining the graph is reduced significantly, in the best case, one operation, at the cost of increased effort in computing the query, in the worst case, O(n), where n is the size of the graph.

Since ACOS supports both on-line ontology construction and on-line ontology query, it needs to strike a balance between the effort for maintaining ontology and that for using ontology. We believe that a non-compressed graph representation can provide such a good trade-off.

To check whether a subClassOf, superClassOf, instanceOf or sameClassAs relationship holds between two classes in an acyclic graph that contains multiple kinds of relationships like subClassOf, sameClassAs and instanceOf, we use grammar-guided analysis method:

sameClassAs(C1, C2) ::= sameClassAs(C1, C2)
| sameClassAs(C1, C3) ^ sameClassAs(C3, C2)
| sameClassAs(C2, C1)
subClassOf(C1, C2) ::= subClassOf(C1, C2)
| subClassOf(C1, C3) ^ subClassOf(C3, C2)
| sameClassAs(C1, C3) ^ subClassOf(C3, C2)
| subClassOf(C1, C3) ^ sameClassAs(C3, C2)
superClassOf(C1, C2) ::= subClassOf(C2, C1)
instanceOf(I1, C1) ::= instanceOf(I1, C1)
| instanceOf(I1, C2) ^ subClassOf(C2, C1)

It is noted that we use Relation and Relation to denote the relation that we want to query and the relation that is denoted in the ontology file respectively. From the grammar above we can also see that Relation sameClassAs satisfies transitive law, commutative law; Relation subClassOf satisfies transitive law.

The consistency of ontology system maintained by ACOS is enforced by checking a set of rules that are given below:

SubClassOf(C1, C2) ^ subClassOf(C2, C1) = false
SameClassAs(C1, C2) ^ subClassOf(C1, C2) = false
Class(CI1) ^ Instance(CI1) = false

The consistency-check algorithm can be described as follows:

Input: ONTS (Ontologies involved in an assert or retract action)
Output: TRUE, if the action won't result in an inconsistency; FALSE, otherwise.

Step 1. Compute the transitive closures of import relation that has ONTS as starting points.
Step 2. Compute the combination of pairs of entities that occur in these closures.
Step 3. FOR each pair in Step 2, check the consistency-check rules given above. IF one of the rules fails, terminate the algorithm and return FALSE.
Step 4. Terminate the algorithm and return TRUE.

At Step 1, we decide the ontologies that may be affected by the change, this can be done at an effort of O(p), where p is the number of ontologies. The computation of pairs and the application of consistency-check rules at Step 2 and Step 3 involve a two-level loop, which needs an effort of O(n×n) ×M, where n is the total number of entities in the ontology system and M is the effort of computing subClassOf(C1, C2) and sameClassAs(C1, C2). Since O(p) << (O(n×n))×M, we have an overall computational complexity of the algorithm: (O(n×n)) ×M.

5 Community Orientated Ontology Management

Once an ontology system has been created by one user using the methodology described above a key issue is how should it be maintained and managed. Previous work has taken the view that a particular user should continue to own and be responsible for all updates to the ontology. This view does not correspond to reality.

Ontology development is a communal activity, and it is an ongoing communal activity; we have come to this conclusion by observing the way in which standards (ontologies are standards) are developed and controlled by committees over a period of time. The membership of the committee is fluid, but controlled. A chairman may or may not have control of the committee. In short, the way in which an ontology should be managed is dependent on the use and community that the ontology has.

In ACOS, every ontology is associated with an owner, a user group, and a subscriber group. The owner is the user who created the ontology. They specify the access rules to their ontology and manage the user group. The user group of an ontology are the owner's trusted users, to whom the owner can give write permissions to create a team development environment. The subscriber group is those who subscribe the ontology as their interface ontology. A subscriber can talk to another subscriber using the terms in its interface ontology. Often, subscribers are the owner or a member of user group of the ontology.

After building the links between the ontology connection graph and user group, a collaborative development environment is established. To go a step further, we have built a flexible access control mechanism and rating system to create a community-oriented co-operation environment. The flexible access control system balances between on the one hand keeping an ontology internally consistent, which is done by maintaining a tight change control system, administered by a small group of people, and on the other hand maintaining the ontology's usefulness by allowing any user who has an interest to initiate changes. We have used the sort of concepts that power the management of the "Slashdot" web site to manage ontology services, however we have introduced some new concepts to cope with the fact that the resources that we are managing here are persistent and have different uses and significance & value.

5.1 Calculating Ontology Ratings

Ontology owners specify flexible criteria for accessing the ontology. Those "good users" who satisfy the criteria can access a ontology even though they are not a member of that ontology's user group. We believe this approach can maintain

ontology's quality without restraining the ability of user to contribute to that resource. Additionally this has the benefit that the ontology may no longer the exclusive property of its contributor. In some cases the ontology contributor may wish to maintain exclusive control of the resource; for example a top-of-the-chain retailer may have the primary objective of forcing its suppliers to conform to its own standards and may simply want an ontology server as an open exchange control mechanism. Alternatively a large company may wish to create a standard to develop a market for exploitation through service products; in this situation its customers may be reassured by the fact that the ontology supplied now has a life of its own.

Consistency checking is applied prior to all the ontology modifying requests. Those requests that will cause inconsistency are not acted upon. Ontologies and users are rated, the ratings of them are connected to each other and are the foundation of flexible access control system. Ontologies will be mainly rated according to their popularity. Let S be an ontology system, O be an ontology, T be a transitive closure (following the relation "imported-by") of O in S, the popularity of O in S is computed as the sum of numbers of subscribed users in T. Figure 2 shows an example of an ontology connection system.

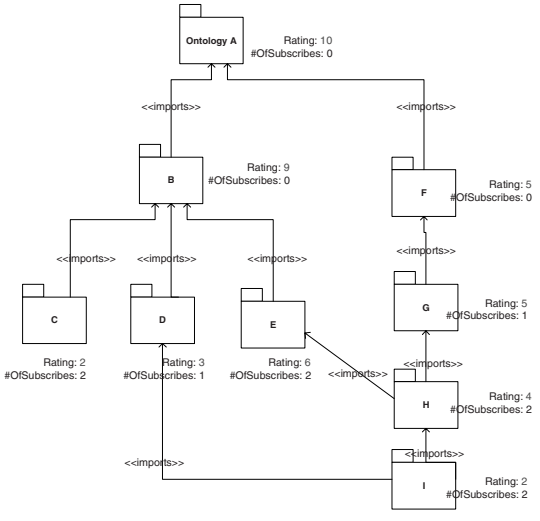

Fig. 2. Ontology Connection System

The calculation process is illustrated in Table 1.

We take the rating calculation of ontology E for example. The ontology E is directly imported by ontology H, and imported by implication into ontology I. Therefore the rating of ontology E (E_k) will be: ontology E's number of subscribers + ontology H's number of subscribers + ontology I's number's subscribers = 2 + 2 + 2 = 6.

From the example, we can see the rating of an ontology will represent the popularity of that ontology, i.e. some vocabularies in the ontology can be interpreted by six users.

Table 1. A rating calculation illustration

Ontology	Full imported set	# of subscriber	rating
A		0	10
B	A	0	9
C	A,B	2	2
D	A,B	1	3
E	A,B	2	6
F	A,	0	5
G	A,F	1	5
H	A,B,E,F,G	2	4
I	A,B,D,E,F,G,H	2	2

5.2 Calculating User Ratings

User ratings are based upon the level of use of the ontologies (ontology rating) that they own and their previous actions. These actions involve asserting and retracting predicates in an ontology. The more successful an action is, the higher rating the user will get; and vice verse. The degree of success for assertion is characterised by a real number that is in proportion to the rating of the ontology being modified. The rationale here is that the higher the rating of an ontology, the more popular it is among the users and it will require more expertise to add new knowledge to it. The degree of failure is in reverse proportion to the rating of the modified ontology and having the nature of the failure as a factor; the explanation is that an agent should receive higher penalty if it fails to add new knowledge in a less popular area. Also, failing to assert an already-implied predicate should receive less penalty than failing to assert an explicitly-exist predicate.

Since retracting a predicate does not affect the consistency of an ontology, the degree of success/failure is not in proportion to the rating of modified ontology. However, it does have the nature of failure as a factor: failing to retract a non-existent predicate will receive heavier penalty than failing to retract an implied predicate.

As updating ontology is considered as a critical activity, thresholds are set up as entry criteria that allow user to undertake the activity; only when the user's rating is higher than the threshold can it assert or retract a predicate.

6 Evaluation

6.1 Critique of Consistency Checking

We discussed, in Section 4, that the overall computational complexity of the algorithm is $(O(n \times n)) \times M$, where n the total number of entities in the ontology system and M is the effort of checking subClassOf(C1, C2) and sameClassAs(C1, C2). The key factor here is M whose computation effort depends on the representation of the graph, i.e. linked format or compressed format. In the former case, which is also our choice, the computation effort is O(n) in the worst case and O(log(n)) on average.

6.2 Critique of Rating System

We have implemented & reported a straw man rating system with the intention of illustrating the principle of developing community administered resources. We have not considered the following factors :

- the computational cost of calculating ratings as part of the management system;
- the relevance and effectiveness of the heuristic for determining ratings

Further investigation is needed to validate the appropriateness of the algorithms adopted.

6.3 Comparison with Other Ontology Servers

Although there are many knowledge repositories since Knowledge Engineering established itself as field in 1970s, most of them were developed as knowledge Bases for expert system applications [3]. The knowledge embedded in these repositories are normally in fine-grained, understood only by few domain specialists and not for the purpose of sharing among general audience. Strictly speaking, according to FIPA definition, these kinds of knowledge are not called *shared ontology* [6].

Ontolingua server [18] is such kind a knowledge repository management system that has been running since 1995. It is a tool that supports distributed, collaborative editing, browsing and creation of Ontolingua ontologies. The ontolingua ontology uses the representation languages of both Ontolingua Frame Ontology [19] and KIF, which is a wide spectrum language capable of representing fine features of concepts. The ontolingua server supports ontology inclusion and circular dependencies [7]. Its consistency-check capability, however, is restricted to the functions similar to database schema checking. For instance, "all slots, slot values, facets and facet values are checked to make sure that they conform to the constraints that they apply (e.g., domain, range, slot value type, and cardinality constraints)" [7]. There is a reasonable choice, given the complicated knowledge structure that Ontolingua server supports; an automatic, on-line, complete consistency-check would be very difficult if not infeasible. The way Ontolingua server ensures knowledge base consistency is through human intervention, where "when someone has an ontology that they believe is ready for others to use, they can choose to publish their ontology in the ontology server. After the ontology has been *approved*, it becomes available to anyone with access to the ontology server through the library" [7]. In other words, the consistency checking is done in domain experts' heads. In comparison, ACOS supports ontologies that can be shared by a wide range of software agent communities and facilitate on-line ontology construction, consistency-check and use.

Driven by the needs and visions for Semantic Web, where ontology server is believed to be a key component [8], there is a surge of work related to ontology server development that is either under the way or in plan [20, 21, 22]. OntoRama [20] is an ontology editor for WebKB that is a web accessible Knowledge Base containing a type hierarchy and knowledge statements. The Knowledge Base supports a set of selected ontology constructors including subtypeOf, partOf, memberOf, urlOf, etc. The main functions of OntoRama include search, compare and modify WebKB ontologies [24]. At current stage, OntoRama does not support consistency-check and

cross-ontology query; as a result, its capability of supporting on-line, collaborative ontology construction is restricted.

UKOLN [21] is an initiative to develop an ontology server for Agentcities.NET project. It proposed that a registry be established on UKOLN agent platform that provides a publication environment for the disclosure of metadata vocabularies and customised application-specific profiles of these vocabularies. As latest as in October 2002, it is still at research and investigation stage, such as investigation of JADE as agent platform, ontology constructors, functions of ontology services, etc. [21].

DOGMA [22], a new ontology server research project, is funded for 5 years by the Vrije Universiteit Brussel. The mission of its ontology server is to "assist the gathering and incremental growth of ontoloiges" [23]. The main novelty of the proposed ontology server, that is in line with the rationale of ACOS, is stated as "the use of the extremely simple structures should – we conjecture – achieve at least a degree of scalability not possible with more complex representations hitherto used in the literature and practice; this extensional approach will naturally lead to very large sets, thereby moving the issue to matters of ontology organisation, rather than of representational power or sophistication" [10]. The proposed ontology model consists of 5 basic elements: context, terms, concepts, roles and lexicons; constraints and derivation rules are intentionally left outside the ontology. At this stage, they are still experimenting with the ontology model and an early (and incomplete) version of ontology server is being implemented. In the first prototype they are building, consistency-check and user control is not included. DBMS, in particular, MSQL server 7.0, is used to store the ontology [23]. Table 2 summarises the features of these ontology services.

Table 2. Comparison of ontology services features

	ACOS	**FIPA Ontology Service**	**Ontolingua**	**OntoRama**	**UKOLN**	**DOGMA**
Knowl. Rep	Simple	Complex	Complex	Med	Under dev.	Medium
Import Mechanism	Yes	Yes	Yes	No	Not mentioned	No
Manner of Consistency Checking	Auto	Open	Human Approval, simple data type checking	Not mentioned	Not mentioned	Not mentioned
Ontology Repository Platform	Jena + DAML +OIL	Open	Ontolingua Server	Jena + RDF	Under dev.	DBMS
User Access Control	Yes	N/A	Yes	No	Not mentioned	No
Managemnt Provision	Comm unity	N/A	Central	Central	Central	Central
Status	Rolled out 2002	Spec.0/08/ 2001	Rolled out in 1995	Rolled out 2000	Under dev.	Being developed

7 Conclusion

The description of ACOS given above discloses a novel system for inferencing over ontologies and managing collections of them. We have discussed the usefulness and utility of ontology servers in engineering systems that operate in the Agentcities environment and that will operate in the Semantic Web in the future.

In the future we intend to investigate the issues of data & structure management and the impact and implications of the strategies that are adopted with respect to these issues. We will also examine how we can handle the uncertainty and ambiguity that arise within knowledge sources developed and used by groups of humans [25]. With respect to technical issues we intend to investigate transactional mechanisms that can be used to manage and resolve knowledge base conflicts in a real time environment and the implementation of a canonical web service interface to the server.

References

[1] *Willmott, S. and Burg, B. and O'Brien, P,.* "May your Information Service Live in Interesting Times: Engineering Dynamic Service Environments", In proceedings 3rd International Conference on Enterprise Information Systems (ICEIS2001), Setubal, Portugal, 7–10th July 2001. July, 2001.

[2] P.R. Cohen and E. Feigenbaum (eds.): "The Handbook of Artificial Intelligence", Morgan Kaufman, 1982.

[3] C.K. Chang, (ed.): "Handbook of Software Engineering and Knowledge Engineering", World Scientific Publishing Co., 2002.

[4] Tim Berners–Lee, James Hendler and Ora Lassila, "The Semantic Web, A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities", Scientific American, May, 2001.

[5] http://www.daml.org/2001/03/daml+oil-index.html

[6] Foundation for Intelligent Physical Agents, FIPA Ontology Service Specification, http://www.fipa.org/specs/fipa00086/XC00086D.html

[7] A. Farquhar, R. Fikes and J. Rice, "The Ontolingua Server: a Tool for Collaborative Ontology Construction", in Proceedings of the 10th Knowledge Acquisition for Knowledge-based System Workshop, Baniff, Canada, Nov., 1996.

[8] R. Agrawal, "Making Semantic Web Real: Some Building Blocks", invited talk, in Proceedings of International Workshop on Semantic Web, Hawaii, 2002.

[9] The Slashdot community portal. www.slashdot.org

[10] Spyns P., Oberle D., Volz R., Zheng J., Jarrar M., Sure Y., Studer R. & Meersman R., "OntoWeb – a Semantic Web Community Portal", in Karagiannis D. & Reimer U., (eds.), in Proceedings of the Fourth International Conference on Practical Aspects of Knowledge Management (PAKM02), LNAI 2569, Springer Verlag, pp. 189–200.

[11] http://www.daml.org/2001/03/daml+oil-ex.daml

[12] Guarino, N., "Formal Ontology in Information Systems", in N. Guarino (ed.) Proceedings of FOIS'98, Trento, Italy, June, 1998.

[13] Gruber, T.R., "Toward Principle for the Design of Ontologies Used for Knowledge Sharing", International Journal of Human and Computer Studies, 43(5/6): 907–928.

[14] A. Borgida, R.J. Brachman, D.L. McGuinness, and L.A. Resnick, "CLASSIC: A Structural Data Model for Objects", in Proceedings of the ACM SIGMOD International Conference on Management of Data, pp. 58–67, Portland, OR, 1989.

[15] P. Compton and R. Jansen, "Knowledge in Context: A Strategy for Expert System Maintenance", C.J. Barter and M.J. Brooks, in Procs of the 2nd Australian Joint Artificial Intelligence Conf, pp.292–306, Adelaide, Aus, Nov., 1998.
[16] P. Compton and R. Jansen, "A Philosophical Basis for Knowledge Acquisition", Knowledge Acquisition 2(3), pp. 241–258.
[17] R. Agrawal, A. Borgida, H.V. Jagadish, "Efficient Management of Transitive Relationship in Large Data Bases", SIGMOD 1989.
[18] http://www-ksl-svc.stanford.edu:5915
[19] T.R. Gruber, "A Translation Approach to Portable Ontology Specifications", in Proceedings of the 2nd Japanese Knowledge Acquisition for Knowledge-based Systems Workshop, Kobe, 1992.
[20] http://www.ontorama.org/doc/intro.html
[21] An Ontology Server for Agentcities.NET, http://www.ukoln.ac.uk/metadata/agentcities/InterimReport.htm
[22] Ontology Server Research, http://www.starlab.vub.ac.be/research/dogma/OntologyServer.htm
[23] G. Zhao, J. Zheng and R. Meersman, "An Architecture Framework of Ontology Development and Deployment", in Proceedings of the E-Society IADIS International Conference, 2003.
[24] P. Martin and P. Eklund, "Knowledge Indexation and Retrieval and the World Wide Web", IEEE Intelligent Systems – Special Issue on "Knowledge Management and Knowledge Distribution over the Internet", July, pp. 18–25, 2000.
[25] Y. Li, H. Yang and W. Chu, "Clarity-Guided Belief Revision for Domain Knowledge Recovery in Legacy Systems", in Proceedings of the 12th International Conference on Software Engineering and Knowledge Engineering, Chicago, USA, Jul., 2000.