A Name-Space Context Graph for Multi-Context, Multi-Agent Systems

Narinder Singh
Computer Science Department
Stanford University
Stanford, CA 94305
singh@cs.stanford.edu

Omar Tawakol
Computer Science Department
Stanford University
Stanford, CA 94305
omart@cs.stanford.edu

Michael Genesereth
Computer Science Department
Stanford University
Stanford, CA 94305
genesereth@cs.stanford.edu

Abstract

Sharing knowledge across a community of users demands agreement about the meaning of symbols. There are two important problems that may arise. First, there is *incompatibility*, where different communities can use different symbols to mean the same thing. This is not as serious as the second problem of *inconsistency*, where the same symbol can be used by different members of a community with different meaning. This problem is especially apparent in multi-agent communications where different agents have different conceptualizations about the world. When agents express these conceptualizations in a communication language, they cannot interact unless they assume a common name-space.

In this paper we propose a name-space context graph (NSCG) for the management of multiple name-spaces. The context graph allows knowledge to be exchanged between agents in separate naming contexts. In order to facilitate this exchange the NSCG performs efficient translation of symbols from one context to another, thus providing each agent with the illusion that there is only one globally unique naming context. This method of handling contexts is general enough for any language, it is not tied to the notion of an address space, it works equally well with declarative languages as well as procedural languages, and it does not require reference to the definition of symbols. Furthermore, the NSCG is suitable both for composing larger domains out of smaller ones in a layered manner or simply for allowing interprocess communication over several domains without name clashes.

Introduction

The explosion of available services on a widely heterogeneous environment such as that provided by the Internet has exposed the weakness in current approaches to interoperability. In order to address this problem, researchers in the ARPA-sponsored Knowledge Sharing Effort have been using knowledge sharing technology to facilitate automated interoperation. This thread of research suggests that programs (called agents) can interoperate anonymously using an Agent Communication Language (ACL). The issue of anonymous interoperation is discussed in (7).

The vision of anonymous interoperation using an agent communication language assumes that agents will coordinate activities based on the content of their communications. This use of a communication language assumes that some receiving agent will be able to understand the meaning of an expression independent of the sender (agent independent semantics). Herein lies the name-space context problem: agent independent semantics requires all agents to agree on a common name-space. This is an unrealistic assumption when one considers the potential interactions at the scale of the Internet. A more realistic assumption is that within a given context, agents would agree on what a symbol means. Thus, the meaning of a symbol would be tied to the context the symbol is communicated with respect to. A name-space context allows agents at the scale of the Internet to partition the global name-space into smaller more manageable name-spaces. Furthermore, communities can dynamically evolve by creating new contexts out of old contexts by using the NSCG.

This proposed methodology is simple since name-space resolution is done without any reference to the meanings of the symbols, yet it is powerful in that it allows agents to communicate using agent independent semantics.

It is useful to evaluate the NSCG with respect to several key issues. Does the naming scheme assume by default that the same terms in different domains match, or is the opposite assumed? Is the naming scheme mainly useful for context resolution in a reasoning system or is it mainly for use as a mechanism embedded in a procedural language? Does the naming scheme scale?

On what basis are symbols used local to a context? The presence of a symbol at a node signifies an agreement between all agents who use that symbol. The NSCG allows agents to specify that certain symbols belong in a given context. The NSCG, however, does not dictate how to specify what the symbol represents (i.e. we are not concerned with how a symbol is defined, we only want to know that such a definition serves as a basis for agreement). A complementary mechanism

must be used to define the meanings of symbols, e.g., a dictionary, lexicon or ontology authoring tool.

The NSCG provides a simple formalization for asserting the identity of symbols in different contexts. This formalization is simple enough to be used for procedural languages as well as declarative languages, though we focus on the latter application in this paper.

Current Approaches

The simplest solution to the naming scheme problem is to use globally unique names. With globally unique names, the problem of naming conflicts disappears at the cost of user- friendly naming. It is difficult to create groups with new vocabularies if all names are globally unique. Whenever a symbol has already been used, a user is forced to come up with a new symbol, and thus the names will become longer and less familiar. This problem becomes apparent in large distributed systems where different processes reuse the same terms with different conceptualizations. As soon as these processes interoperate they effectively form a common name-space and the threat of naming clashes arises.

Name-Spaces as a Programming Construct

The problem of name-space management has been dealt with in a variety of manners in different programming paradigms including the use of file inclusion in C, ML style modules, Lisp style packages (14), and the newly adopted name-space mechanism in C++ (4). These solutions, however, are language dependent and do not offer mechanisms for globally unique naming at the scale of the Internet where different programs are expected to interoperate across heterogeneous environments. Furthermore, these solutions may not be compatible with reasoning systems.

Contexts in AI

In AI, contexts have been proposed as a means of defining the truth of a sentence relative to a situation. Many concepts that a reasoner uses are limited to a particular context. Such a context may involve many assumptions which are relevant to the truth of a proposition made in a particular context. McCarthy and Buvac's "Formalizing Contexts" (12), develops this notion of context which allows axioms for static situations to be "lifted" to more dynamic contexts where situations change. To this end, the `ist(c,p)` formula is introduced so that a proposition, `p` is true in a context `c`.

According to McCarthy,

> Introducing contexts as formal objects will permit axiomatizations in limited contexts to be expanded to transcend the original limitations. This seems to provide AI programs using logic with certain capabilities that human fact representation and human reasoning possess.

Although this notion of context has influenced the discussion of naming contexts in this paper, our notion of a name-space context has a different scope. Whereas the sentence `ist(c,p)` is concerned with the truth of a proposition in a context, the NSCG is only concerned with the identity of a symbol in a given context.

Our notion of a context can be used by a logical reasoning system, however, we provide no machinery for manipulating contexts as first order objects. Instead, we provide, a mechanism for defining the equivalence of symbols in different contexts. An example of the difference between the McCarthy/Buvac notion of a context and our notion of a name-space context is given below:

```
ist(context-of("Sherlock Holmes stories"),
    "Holmes is a detective")
```

This asserts that "Holmes is a detective", and it asserts it relative to the "Sherlock Holmes stories" context. In contrast, the NSCG can only be used to specify the identity of the symbols (`Holmes`, `is`, `a`, `detective`). For example, the NSCG could specify that in the "Sherlock Holmes stories" context the symbol "Holmes" is distinct from the symbol "Holmes" in the "US legal history" context. We can use the NSCG with a first-order logic system to specify that "Holmes is a detective," where all the symbols in this sentence are local to the context "Sherlock Holmes stories." The NSCG assures us that the symbol "Holmes" in this context will not be confused with the symbol "Holmes" in some other context. Furthermore, the NSCG can be used to define the inheritance of symbols based on the structure of the graph, e.g., it can specify that the "fictional characters" context inherits the "Holmes" symbol from the "Sherlock Holmes stories" context and not any other context.

The NSCG model can be limiting, e.g., there is no calculus of contexts (the NSCG cannot be used to talk about the disjunction, negation, etc. of a collection of contexts), and it is not possible to define mappings that are more complicated than identities (the NSCG cannot be used to define conditional mappings between a symbol in one context and a symbol in another context). However, such mappings can be provided by mechanisms complimentary to the NSCG, e.g., axioms in first-order logic.

Overview of the NSCG

The NSCG is concerned with supporting the communication of information between agents in different contexts. There is a default root context, which forms a minimal core for communication between all agents (all agents agree to the meaning of symbols in the root context). The NSCG supports the creation of new contexts by building on existing contexts (union operation), and permits overriding definitions of inherited contexts (difference operation).

The NSCG is a rooted directed acyclic graph. A node is a name-space context and an edge is an ordered pair of nodes <parent, child>. Each node has a

set of symbols that are local to it. Furthermore, each node (except the root) specifies its parent nodes. A child node "inherits" the symbols that are local to a parent node. Thus, a symbol local to a parent node can be referred to within a child without ambiguity (the symbol at the child node is identical to the symbol at the parent node). If more than one parent has the same symbol local to it, then the child must select one of the parents to inherit from. Alternatively, a child may override a local symbol of one or more parents by making the symbol local to it (shadowing).

A symbol local to a context is a decontextualized symbol (e.g., `pound`). A contextualized symbol explicitly indicates its context, and is of the form `symbol@context`. A contextualized symbol can be unambiguously referred to in any context. A decontextualized symbol can map into a contextualized symbol by concatenating the context to the symbol (e.g. `pound@currency`).

As an example, the symbol `pound` in the naming context `mass` is `pound@mass` in a global context while the symbol `pound` in the naming context `currency` is `pound@currency`. A naming context `gym` is free to have the two parents `mass` and `currency` as long as it specifies the identity of the symbol `pound` to be one of `pound@currency` or `pound@mass` (alternatively, it could also choose to have a local identity for `pound`).

In designing a naming scheme there are generally two approaches: 1) the same symbol in different contexts are assumed to be distinct by default, and 2) the same symbol in different contexts are assumed to be identical by default. The NSCG permits a compromise between these two extremes, depending on the structure of the graph and the symbols local to a node (which define what symbols are inherited):

- If two contexts are part of an ancestor-descendant relationship, then a symbol in the descendant context is identical to the same symbol in the ancestor context.
- If there is no directed path between two contexts, then a symbol in one context is distinct from the same symbol in the other context.

The NSCG scheme has the following useful properties:

- Symbols need only have a unique definition with respect to their naming context.
- Two different symbols in different contexts can be defined to be equivalent.
- A symbol in one context can be mapped into a global context or into any other context.

These properties allow the communication of concepts from one context to another without the threat of incompatibilities and inconsistencies. Due to the simplicity of the graph, implementations of this formalization can be easily maintained and are scalable at the level of the Internet.

Examples of Using a NSCG

In this section we provide some examples illustrating the capabilities of a NSCG. There are two important issues related to a NSCG: 1) the structure of the graph, and 2) communication between different contexts in the graph. We discuss these briefly below.

Modifying the Structure of a NSCG

In this subsection we illustrate the process by which a NSCG may evolve. In the beginning we are assuming that graph consists of a single root node, and as the needs change nodes and edges are added to the graph.

Example 1: Universal agreement on symbol definitions– using a single node. A new shoe company Reebok is formed. A node N_r is created for it in a NSCG, and the corporation defines the official words and their meanings for the organization to use. The symbols used by Reebok are all defined to be local to the node N_r in the NSCG (they are not inherited). All communication at Reebok is with respect to the node N_r.

Example 2: Creating a new node to extend the set of symbols. Reebok is growing rapidly, and decides to get outside help for its taxes. The external tax company uses some vocabulary which is not part of N_r. Reebok plans to have an internal tax department in the future and does not wish to include the vocabulary of the tax company in N_r. Instead it adds a child node N_t to N_r. The node N_t defines the vocabulary for tax terms locally (the symbols related to taxation are local to N_t), and it inherits all other symbols from the parent node N_r. The tax department communicates with respect to the node N_t, and all other departments communicate with respect to node N_r.

Example 3: Creating a new node to redefine symbols. Reebok later creates an internal tax department. The new department looks at the existing node N_t and finds that most of the definitions there can be used directly, except for those related to depreciation. Instead of discarding the node N_t from the NSCG, Reebok decides to specialize it by adding a node N_d as a child of N_t. Node N_d defines the differing vocabulary for depreciation. All symbols related to depreciation are now local to N_d, even though the same symbols are also local to N_t. The tax department now communicates with respect to node N_d, and the other departments continue to communicate with respect to node N_r (no one communicates with respect to node N_t now).

Example 4: Creating a new node that inherits from multiple parents. Reebok has been extremely successful in Europe and wishes to expand to the international market. Unfortunately, other shoe companies are not faring as well. LAGear, for one, is in fiscal trouble and agrees to be bought out by Reebok. There is an existing node N_l for LAGear in the NSCG. Reebok creates a new node N_i which has two parent nodes N_r and N_l.

Unfortunately, there are symbols that are local to both N_r and N_l which have different definitions. These

conflicts must be resolved for node N_i. Reebok prefers its definitions in node N_r over the definitions in node N_l for the conflicting symbols. The NSCG is modified so that each conflicting symbol in node N_i is defined to be identical to the same symbol in node N_r.

The NSCG must still resolve the problem of different nodes using different symbols for the same concept. Unfortunately LAGear uses the word `color`, while Reebok uses the word `colour` for the same concept. To resolve this, the NSCG is modified so that the symbol `color` is no longer local to the node N_l. The symbol `color` at node N_l is now defined to be identical to the symbol `colour` at node N_r.

Communication in a NSCG

An important application of the NSCG is in communication between arbitrary contexts. A NSCG provides a partitioning of the set of symbols (the same symbol in different contexts can be distinct, and different symbols in different contexts can be identical). The structure of the NSCG makes some communication convenient (communicating symbols without being explicit about their context), while other communication is more involved (communicating symbols by making their context explicit).

In the remainder of this subsection we illustrate different types of communication between nodes in a NSCG.

Representing Symbols Relative to a Context It is possible to simplify the syntactic form of a symbol when representing it relative to a context. Suppose we wish to represent the symbol $s@c_1$ relative to the context c_2. We can drop the explicit context c_1 if $s@c_1$ is identical to $s@c_2$.

For example, suppose the context `reebok` is a descendant of `kif`. Then, the sentence:

```
(>@kif (size@reebok shoe1@reebok) 9@kif)
```

can be represented relative to the context `reebok` as

```
(> (size shoe1) 9)
```

if:

- `>@reebok` is identical to `>@kif`, and
- `9@reebok` is identical to `9@kif`

In other situations, only some of the symbols in a sentence may have their context dropped. For example, the sentence:

```
(<@kif (size@reebok   shoe1@reebok)
       (size@LAGear shoe3@LAGear))
```

can be represented relative to the context `reebok` as

```
(< (size shoe1)
   (size@LAGear shoe3@LAGear))
```

if:

- `<@reebok` is identical to `<@kif`,
- `size@reebok` is *not* identical to `size@LAGear`, and
- `shoe3@reebok` is *not* identical to `shoe3@LAGear`

Communicating Contextualized Symbols Communication must preserve semantics. Therefore, the identity (meaning) of a symbol $s@c$ must be the same in a sending context c_s and a receiving context c_r.

A contextualized symbol $s@c$ can be communicated from any sending context c_s to any receiving context c_r without transformation. For example, the fact

```
(>@kif (size@reebok shoe12@reebok) 9@kif)
```

can be communicated directly from any context to another, since the owning context of every symbol is explicit (all symbols are contextualized).

Communicating Decontextualized Symbols If a symbol is represented relative to a context, then its communication to arbitrary contexts is made possible by specifying the owning context for every symbol in the fact.

For example, if the fact:

```
(< (size shoe1)
   (size@LAGear shoe3@LAGear))
```

is represented relative to the `reebok` context, then we can communicate it globally to arbitrary contexts by the fact:

```
(<@kif (size@reebok   shoe1@reebok)
       (size@LAGear shoe3@LAGear))
```

if `<@reebok` is identical to `<@kif`.

Communicating Across Contexts Sometimes it is appropriate to communicate a symbol from one context to another. The symbol may or may not be contextualized in the sending context. This may be useful, for example, in communicating between agents, where one agent represents facts relative to context N_i and another agent represents facts relative to context N_j.

We illustrate the process by an example first. The symbols in the following fact are represented relative to context `reebok`:

```
(< (size shoe1)
   (size@LAGear shoe3@LAGear))
```

We wish to transform its representation to be relative to the context `la-gear`. The result is:

```
(< (size@reebok shoe1@reebok)
   (size shoe3))
```

if:

- `<@reebok` is identical to `<@LAGear`, and
- `size@reebok` is *not* identical to `size@LAGear`

The general procedure M maps a symbol s, a source context N_i, a destination context N_j, to a symbol s'. It first contextualizes the symbol s with respect to the context N_i, and then decontextualizes the resulting symbol with respect to the context N_j. The details are presented towards the end of the next section.

NSCG Formalization

A Name-Space Context Graph (NSCG) is a rooted directed acyclic graph. Formally, it is a tuple: $< N, E, r, S, I >$

Where:

- N is the set of nodes in the graph. We sometimes call a node a context,
- E is a set of edges: $< N_i, N_j >$, where N_i, $N_j \in N$, (N_i is the parent),
- r is the root node of the graph,
- S is the set of all symbols. Every context must define symbols from S,
- I is the explicitly defined identity mapping function for symbols at a node.

I specifies the explicit identity mapping between a symbol at a node and the equivalent symbol at another node. Therefore, if $I(s_j, N_j) =< s_i, N_i >$ then $s_j @ N_j \equiv s_i @ N_i$ (i.e., the two decontextualized symbols are equivalent). More formally:

$$I : S \times N \rightarrow S \times N \cup \perp$$

Note that $\perp$ is the undefined symbol, so I need not be complete.

We say that a symbol s is local to a node N_i if $I(s, N_i) =< s, N_i >$.

We place the following constraints on the function I:

- I must be direct, that is if $I(s_j, N_j) =< s_i, N_i >$, then $I(s_i, N_i) =< s_i, N_i >$ (s_i is local to N_i). This is to ensure that I is well founded.
- I must be ancestral, that is if $I(s_j, N_j) =< s_i, N_i >$, then N_i is an ancestor of N_j. A node N_i is an ancestor of N_j if there is a directed path from N_i to N_j. This is to ensure that I only depends on the nodes on the path back to the root node (it cannot rely on the structure of other parts of the graph).
- I must be defined for a symbol s at a node N_k, if N_k has two (or more) ancestors where s is local. For example, if for any two ancestors N_i and N_j of N_k it is the case that $I(s, N_i) =< s, N_i >$ and $I(s, N_j) =< s, N_j >$), then $I(N_k, s) \neq \perp$. This is required to resolve conflicts with inheritance.

We place no other constraints on the function I (nor how it should be defined in a given implementation).

The NSCG defines a rooted DAG of nodes. The purpose of the graph structure is to provide identity mappings for symbols when these are not explicitly specified. This is similar to inheritance in a programming language.

We define the partial completion of the function I as the function I_{pc}. Given the definition of I, and the nodes and edges in a NSCG, the function I_{pc} partially completes the definition of I for a node N_i by taking into account the definition of I for the parents of N_i (inheritance). More formally:

$$I_{pc}(s, N_j) = \begin{cases} I(s, N_j) & \text{if } I(s, N_j) \neq \perp \\ I_{pc}(s, N_i) & \text{if } I_{pc}(s, N_i) \neq \perp \\ & \text{and } < N_i, N_j > \in N \\ \perp & \text{otherwise} \end{cases}$$

The function I_{pc} is used to resolve the identity (meaning) of a symbol in a NSCG. The only service provided by the NSCG is defining the identity mapping between symbols. For example, if $I_{pc}(s_j, N_j) =< s_i, N_i >$, then the NSCG defines the equivalence $s_j @ N_j \equiv s_i @ N_i$.

Given the definition of the identity function I_{pc}, we define the mapping function M, which maps a symbol s, a source context N_i, a destination context N_j, to a symbol s'. Note that the symbols s and s' may or may not be contextualized.

We define the function M as the composition of the functions contextualize C and decontextualize D: $M(s, N_i, N_j) = D(C(s, N_i), N_j)$

The function contextualize takes a symbol s, a source context N_i, and maps it to the contextualized symbol s'. The definition of C is given below.

Case I: s is a contextualized symbol of the form $s'@N_k$

$$C(s'@N_k, N_i) = \begin{cases} s''@N_l & \text{if } I_{pc}(s', N_k) =< s'', N_l > \\ \perp & \text{if } I_{pc}(s', N_k) = \perp \end{cases}$$

Case II: s is a decontextualized symbol

$$C(s, N_i) = \begin{cases} s'@N_k & \text{if } I_{pc}(s, N_i) =< s', N_k > \\ \perp & \text{if } I_{pc}(s, N_i) = \perp \end{cases}$$

The function decontextualize takes a contextualized symbol $s@N_i$ and a destination context N_j, and maps these to a symbol which is to be represented with respect to N_j. The definition of D is given below:

$$D(s@N_i, N_j) = \begin{cases} s' & \text{if } I_{pc}(s', N_j) = I_{pc}(s, N_i) \\ s@N_i & \text{otherwise} \end{cases}$$

Note that in some situations a symbol is mapped from a source context to a destination context without transformation, e.g., when communicating a symbol from a node to one of its descendants (e.g., when none of the nodes in the path to the descendant redefine the identity of the symbol).

Note that the constraints on I_{pc} permit:

- mapping a symbol from one context to a different symbol in another context. For example, $I_{pc}(s_j, N_j) =< s_i, N_i >$ and $s_j \neq s_i$.
- a node N_i to locally own a symbol, even if one of its ancestors locally owns the same symbol (shadowing). For example, $I_{pc}(s, N_j) =< s, N_j >$ and $I_{pc}(s, N_i) =< s, N_i >$ for some ancestor N_i of N_j.
- a node N_i to select a particular ancestor, if more than one ancestor has the symbol local to it (shadowing-import). For example, this is the case

if node N_i has two parents N_j and N_k, where $I_{pc}(s, Nj) \neq I_{pc}(s, N_k)$ and $I_{pc}(s, N_i) = I_{pc}(s, N_j)$ or $I_{pc}(s, N_i) = I_{pc}(s, N_k)$.

Conclusion

We advocate the use of a Name-Space Context Graph as a means of managing name-spaces in multi-context systems. This is particularly useful in anonymous interaction between agents in heterogeneous environments. In such a system where agents interact without knowledge of each other's representations, there must be some way of partitioning each agent's conceptualizations of the world without isolating them. The NSCG achieves this by allowing each agent to specify the conceptualization context that it uses. When interaction is required, the NSCG can be consulted to perform translation from one context to another without making use of the semantics of the symbols (i.e., purely syntactic translation).

Having evaluated the deficiencies of using globally unique names we have shown how the NSCG allows for the same benefits while allowing users to only need unique names within a given naming context. The NSCG also allows the use of incompatible theories in a general theorem prover by separating theories with contexts. Thus, the NSCG provides a simple and flexible mechanism for name-space management.

References

[1] Buvac, S. and Fikes, R., "A Declarative Formalization of Knowledge Translation," Stanford University, 1994.

[2] Buvac, S. and Mason, I., "Mathematics of Contexts," Fundamenta Informaticae, 23:3, 1995.

[3] "C++ Name Spaces," *Dr. Dobb's Journal,* Pennello, Tom, August 1994.

[4] Farquar, A, Dappert, A., Fikes, R., Pratt, W., "Integrating Information using Context Logic,", DRAFT PAPER, KSL, Stanford University, December, 1994.

[5] Finin, T. and Wiederhold, G., "An Overview of KQML: A Knowledge and Manipulation Language," available through the Stanford University Computer Science Department, 1991.

[6] Genesereth, M., Fikes, R., et al., "Knowledge Interchange Format Version 3 Reference Manual," Logic-92-1, Stanford University Logic Group, 1992.

[7] Genesereth, M., Singh, N., and Syed M., "A Distributed and Anonymous Knowledge Sharing Approach to Software Interoperation," *International Symposium on Fifth Generation Computer Systems 1994*, Workshop on Heterogeneous Cooperative Knowledge-Bases, pp 125-139, 1994.

[8] Attardi, G., and Simi, M., "A Description-Oriented Logic for Building Knowledge Bases," *Proceedings of the IEEE,* Vol 74, No. 10, October 1986.

[9] Attardi, G. and Simi, M., "A formalization of Viewpoints," TR-93-062, International Computer Science Institute, Berkeley CA, October 1993.

[10] Gruber, T., "Ontolingua: A Mechanism to Support Portable Ontologies," KSL-91-66, Stanford Knowledge Systems Laboratory, 1991.

[11] Guha, R., "Contexts: A Formalization and Some applications," PhD. Dissertation, Stanford University, Technical Report No. STAN-CS-91-1399.

[12] McCarthy, J. and Buvac, S., "Formalizing Context," Expanded Notes, 1993.

[13] Nayak, P., "Representing Multiple Theories,", NASA Ames Research Center, 1993.

[14] Steele, G., *Common Lisp the Language,* Digital Press,

[15] Wiederhold, G., Wegner, P., and Ceri, S., "Towards MegaProgramming: A Paradigm for Component-Based Programming," Stanford University, Brown University, Dipartimento di Elettronica e Informazione, June 1992.

[16] Wiederhold, G., "An Algebra for Ontology Composition," ARPA and Stanford University, July 1994.