

a Network Inference Engine
dsl.cis.upenn internal project report
(24th October 2003)

Willem de Bruijn
wdebruij@dds.nl

Contents

1	introduction	3
2	design choices	5
2.1	separation of concerns	5
2.2	knowledge representation	6
2.3	NIE locality	6
3	architecture	8
3.1	knowledge representation	9
3.2	inference engine	11
3.3	knowledge layout	12
4	implementation	14
4.1	prolog-based inference engine	14
4.2	basic IO	15
4.3	distributed IO	16
4.4	human computer interface	17
4.5	interacting with the NIE	18
5	related work	19
6	current status and future work	21
7	concluding remarks	23
A	further reading	24
A.1	decision making architectures	24
A.1.1	case-based reasoning	24
A.1.2	model-based reasoning	24
A.1.3	prolog	25
A.2	knowledge representation	26
A.3	XML and XSLt, XHTML and CSS	26
A.4	RDF	27
A.4.1	Tutorials	27
A.4.2	Non Prolog RDF Inference	28

A.4.3	Prolog and RDF	28
A.4.4	Semantic Web	28
A.4.5	tools	29
A.5	other	29
A.5.1	trust management	29
A.5.2	distributed messaging	30
A.5.3	APNets	30
B	software listing	31
B.1	prolog sourcecode	31
B.2	wjdb_prolog_c library	31
B.3	wjdb_list library	32
B.4	rdf dictionaries	32
B.5	human computer interface	33
B.6	other	33

Chapter 1

introduction

This document is supposed to fulfill two functions: it is devised to act both as (1) a user's guide, containing ample technical documentation, and (2) an overview of the rationale used to select certain key features. As such, it is neither a full technical reference, nor a research paper, but merely a quick 'n dirty introduction into the Network Inference Engine (NIE) framework for anyone interested in expanding on the work done so far.

To fully explain the NIE's purpose it is imperative that we first discuss the environment in which it lives. The NIE is the result of a brief introductory study in expert systems tailored to a specific problem, namely that of increasing network adaptivity. The overall research initiative within which the NIE research is embedded is called Application Private Networks or APNets [apn]. APNets tries to overcome certain limiting factors in today's networks by increasing the flexibility of the network stack. For this purpose it projects to replace or supplement the existing network stack, most notably the TCP/IP stack, with more modular, special purpose protocols that are created on the fly from a set of 'mini'-protocols.

Contrary to complex protocols that have to deal with all network issues in all possible situations, TCP can be taken as a prime example of these, modular protocols are constructed from a set of mini-protocols to suit a specific situation. Intuitively, such an approach has many possible benefits: protocol sets are extensible, allowing for addition of new functionality after initial deployment, protocol code can be simplified and transmission overhead can decrease. At the time of writing no actual implementation exists, therefore the argument for modular networks is mostly intuitive.

Adding a level of complexity on top of the modularization, adaptive networks must be able to adapt, i.e. choose network protocols from the input set based on the application's needs and network conditions at runtime. Adaptive networks have, at any time, the following information about their state: available network elements, e.g. availability of the previously mentioned mini-protocols, a current network state and application preferences. The option space can therefore be easily described as a three-tuple: *network elements, network-state, applica-*

*tion preferences*_{*j*}. This three-tuple can be taken as input to a decision making algorithm. For any input, the algorithm supplies as output an optimal set of network elements for constructing a protocol and route through the network.

This decision making part of the APNet initiative is where the NIE comes into play. However, the NIE should be seen as an introductory investigation into the question how such an algorithm should be devised, not a full fledged application. Also, not having access to any of the three variables needed to create an input set, the prototype NIE architecture extends beyond the simple blackbox algorithm to include network status input sensors and a sample set of protocol elements.

The Network Inference Engine as we propose it has a number of features that sets it apart from similar initiatives: (1) There is an explicit distinction between knowledge and the decision making algorithm. (2) Knowledge is expected to be freely available (as in beer). (3) While knowledge might be global, decision making is inherently a biased operation. Therefore, the inference engine is inevitably a user-centric device as opposed to a centralized oracle-like application.

The following chapter 2 discusses the design choices we made when creating a prototype NIE, while chapter 3 presents an architecture based on these choices. In chapter 4 an actual prototype implementation is shown. In chapter 5, some related work is discussed. Chapters 6 and ??, finally, deal with the current status of the prototype and contain our concluding remarks, respectively.

Chapter 2

design choices

Decision making is an integral part of any active system and networks are no exception. However, at present, general purpose networking code is, in most cases, considered part of the OS kernel codebase. Often, the principal kernel optimization vector is performance and many of today's OS kernels indeed are optimized for speed. While this is an acceptable, even preferable strategy for large parts of the operating system's functionality, it does have several major drawbacks. For instance, kernel code is generally written in a very low level language, such as C, which can make updating and extending somewhat difficult. Of more direct impact to us is the fact that all decision making is hardcoded into these low-level routines. Not only will this lead to (1) a bloat in code when the option space increases and therefore a hard to maintain sourcetree, it also (2) limits decision flexibility to a large extent. Naturally, whenever performance *is* the most important concern, this design choice is the correct one. However, in case of network reasoning, where response times do not have to be in the order of nanoseconds, we can safely select a different solution.

2.1 separation of concerns

To circumvent the aforementioned problems, we chose to separate the regular execution engine, which can be written in any language, from the actual knowledge, which is preferably encoded in a meaningful high level language. Applications that separate the programming logic from the knowledge they process are generally referred to as knowledge-based systems or expert systems. There is plenty of information to be found on decision making architectures in general and expert systems specifically. In [Hop00], for instance, Hopgood gives a decent introduction of the field.

The principal advantage of expressing knowledge explicitly, rather than implicitly in kernel routines, is not code maintainability, but reasoning flexibility. In a distributed environment it can be expected that different applications have somewhat different views of the network. Exchanging their views is mutually

beneficial to overcome issues relating to faulty or insufficient data. Acknowledging this fact, distributing network information is currently an active research topic, with a number of promising projects underway, e.g. David Clark's knowledge plane [CPRW03]. Adaptive networks, for one, would benefit from maximizing their knowledge of the state of the networks. Therefore we deemed it imperative to include support for such behaviour in the NIE. However, reasoning with new information is not possible when the logic is hardwired. To allow exchange of knowledge we must have some sort of explicit knowledge representation.

2.2 knowledge representation

Moving knowledge out of the programming code does add a new concern, namely that of choosing a knowledge representation. While we do not deal with a grand vision of a global knowledge plane, interoperability is of importance if we want to maximize our possible data input set. For this reason we have taken the strategy to use industry supported and standardized dataformats whenever possible. A secondary advantage of this strategy lies in that we can exploit existing software packages, instead of having to start from scratch.

2.3 NIE locality

When building any distributed application one must consider where in the network it is to live. A simple choice is to have a central entity that makes decisions based on global knowledge for every participant in the network. Napster, for instance, followed this blueprint closely for connecting producers and consumers of music. This solution suffers from a number of drawbacks. We can disregard the obvious legal implications this network topology had for the napster network. However, scalability and robustness can be problematic in a centralized topology. An obvious answer to this is to use a semi-decentralized network of regular nodes and information servers, such as is used in many of today's peer-to-peer filesharing applications. While this does solve most of the practical issues, it adds complexity and reduces transmission efficiency. From a practical point of view, therefore, one would prefer to disregard both of these options.

Leaving these practical concerns for now, we state that the best solution is to have the decision making process work on behalf of users. When reflecting on the current state of networking, decision making is done by the people that pose the application requirements. Internet service providers and end-users, for instance, can have completely different requirements for their applications. With the same network state and available network elements, their reasoning engines should therefore give completely different output. A centralized solution, however, will not. Since the decision algorithm is static it cannot take into account their biases. One can add both actor's rulesets to the engine, or prefer one over the other, but doing so implicitly adds conflict resolution tactics. Instead, conflict

resolution with regard to resources in the network is a separate affair altogether, and should therefore be placed outside of the decision making process.

Practical issues push us away from using a centralized approach. Explicitly allowing a biased reasoning process, too, guides us to a more decentralized user-centric approach. This still leaves many possible architectures open to discussion. For instance, a single centralized agent per actor, a platform for mobile agents to crawl through the distributed knowledge plane in search for information or a functional approach, where the decision making architecture is only instantiated when needed.

As of now, we have not made any deliberate choice among these options. Out of practical necessity, the current prototype system resembles the first approach most closely, although it can also be instantiated in a functional manner.

Chapter 3

architecture

The architecture resulting from the previously described design choices, depicted in figure 3.1 consists of a general purpose inference engine, its internal ruleset, and various input and output options. For data exchange with other NIEs a formally defined knowledge base syntax is also needed.

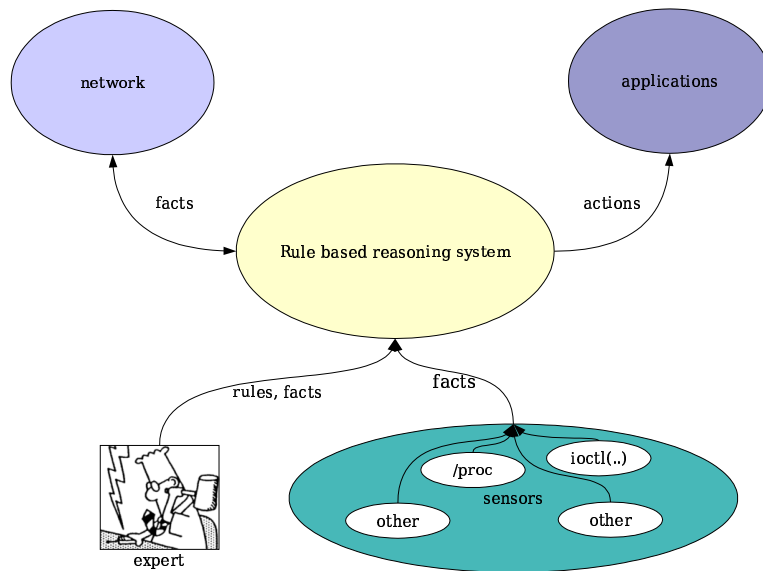


Figure 3.1: NIE architecture overview

3.1 knowledge representation

Knowledge about the network will have to be obtained from many different sources, such as */procfs*, *SNMP* and human input. The complexity of the knowledge obtained from the various sources can differ a lot. Information obtained from the operating system, such as available network devices and routing tables can be seen as low-level information. It is largely unstructured and contains little correlation. More useful is statistical data, which is still unstructured, but identifies correlation between resources in the network. Lastly, and most high-level, is the structural information, most often described in rules, that ties the independent sources together. The way these types of information will be fed into a knowledge repository differ, too. We introduce the term sensors for sources that feed uncorrelated information to the knowledge repository. Sensors can be subdivided into two broad categories: resource sensors, measuring network conditions, and application sensors, which extract application preferences such as sustained packet rate or destination host. Additionally, we will define an expert to be a source of structured information. While sensors are basically dumb devices, experts have to have a broader view. They can be humans or AI agents, such as machine learning algorithms.

The knowledge base has to have a representation that can encode information of various levels of complexity. The more general the knowledge representation is, the more informative the knowledge contained. High level representation languages are abundant and selection can therefore be based on secondary motives. Since we are creating what is to be a distributed knowledge space and interoperability with existing sources of information is preferable, we decided to find a widely supported internet standard. Roughly, two grammars are commonly in use for network data on the internet. Firstly, the ASN.1 syntax is generally used for exchange of network management information, for instance by the SNMP framework. Second, XML is used to encode many information sources, from webpages (XHTML) to newslings (RSS). In theory, both are widely supported and can be used. However, because XML clearly separates syntax from semantics it has an added benefit. By creating dictionaries of terms, XML formatted data can be given meaning. For this purpose it is being used as the predominant (although not the only) serialization of a knowledge exchange format called the Resource Description Format [citerdf-home](http://www.w3.org/2000/01/rdf-schema#).

RDF was developed for a problem similar to the one we are dealing with. Many of today's online resources are written in standardized syntaxes, allowing digital processes to manipulate those resources. However, lacking clearly defined terminology, the information embedded in these syntaxes is only of use to humans. If we could get machines to parse the information in a meaningful way, on the other hand, many everyday tasks could be automated. An initiative called the Semantic Web [BLHL01] tries to move information encoding away from human-centric natural languages to clearly defined machine-usable semantics. The main output of the Semantic Web initiative so far is RDF: a simple mechanism to describe relationships between resources in the network. That it is tailored especially to use in networked based environments shows from the ex-

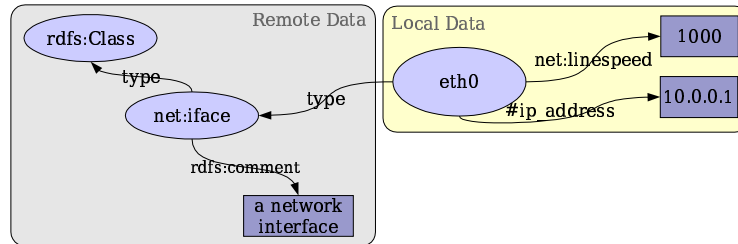


Figure 3.2: Example RDF graph

tensive use of URIs to identify resources. An RDF statements consists of three variables : an object, a predicate and a subject. Together this triple defines a relationship, for example:

```
subject = http://splash-snap.sourceforge.net
predicate = dc:creator
object = mailto:wdebruij@dds.nl
```

connects two networked resources through the *dc:creator* predicate. Although it may not seem this way to people unaware of XML syntax, this too is a URI when the dc is linked to a resource itself. In our example dc is an abbreviation of <http://purl.org/dc/elements/1.1>, an online XML file containing an RDF description of, among others, the *creator* predicate.

From the previous example can be seen that RDF statements rely on dictionaries to give meaning to relationships. One of the main advantages of RDF in our case is that it does not rely on static dictionaries. Instead, any URI can describe an RDF dictionary. Through use of XML as its main serialization syntax, RDF allows using existing software packages for raw language parsing. By adding the notion of dictionaries and relationships, it can be used to express diverse types of meaningful statements. There is a lot of information available online on both XML and RDF, links can be found in the further reading section A. One interesting development in the field is the emergence of a cousin of RDF called Ontology Web Language (OWL) [AvH03] that extends

the language beyond its current constructs and defines special purpose language subsets with possibly desirable characteristics. At the time of writing, however, too little information about OWL is made available to choose it over RDF for our knowledge representation.

3.2 inference engine

To process its acquired knowledge, a decision making architecture needs some sort of inference engine. The brain of the system, the engine must be able to deduce conclusions from a base set of statements. The simplest and most rigorously defined type of inference engine is a logical parser. Taking as input propositions or predicates, it can output formal deductions. A widely used logic parser is the PROLOG programming language. Contrary to procedural languages, prolog code does not define the execution of a number of actions. Instead, it contains logical statements known as Horn clauses: a subset of first degree predicate logic, where the lefthand side of the statement can only contain a single predicate or fact. When clauses are regarded as goals, PROLOG is an execution engine that tries to fulfill these goals. It uses a backtracking algorithm to instantiate the variables needed to fulfill a goal. As such it belongs to the group of backward-chaining problem solvers as opposed to forward-chaining problems solvers, which freely generate new statements from a base set of statements. Prolog can be preferred as inference engine because of its relative simplicity and rigorous formal syntax. For the NIE, we did select prolog as the main parser, mainly because of its track record in similar applications.

Rule based parsers can suffer from a few problems. Since they are in essence formal logic processors, they lack intrinsic support for such issues as lacking information to fulfill a goal, having multiple solutions ($\{1\}$) or none ($\{\}$), reasoning with uncertain information or with fuzzy information. Many alternative approaches exist, to name a few: probabilistic reasoning, machine learning, pattern matching, fuzzy logic parsing and model based reasoning. In a later stage we expect it to be useful to scrutinize each of these alternatives, as well as the simple rulebased approach, to create a more powerful engine. However, by separating the knowledge from the inference engine's internal representation and by employing a user-centric approach, choosing an inference engine is of small importance. It is even foreseeable that different types of engines coexist in a network. Selection of the engine then becomes part of a user's bias, not the global invariant.

A last remark about the inference engine and various extending technologies: some artificial intelligence techniques that might be used as inference engines can also be moved outside of the main execution loop and, instead, act as external experts. Mentioned in the previous section, machine learning algorithms, but equally pattern matching and datamining algorithms can be used to update the ruleset of the inference engine and the knowledgebase from the outside. This hybrid approach would keep the strongpoints of both techniques, while removing many of their individual drawbacks.

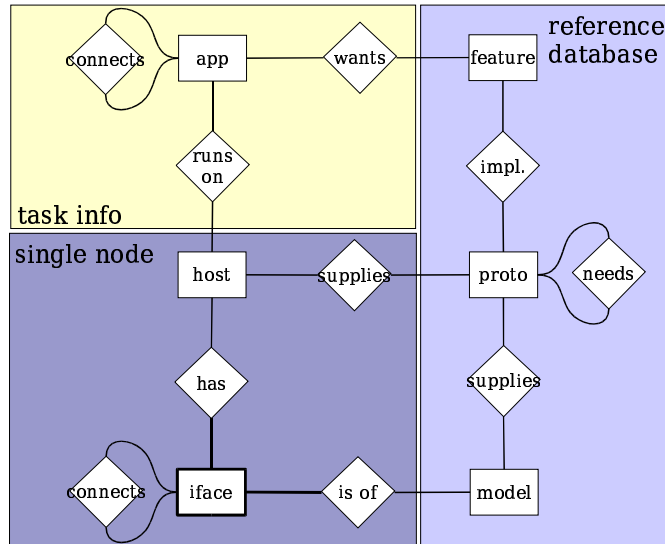


Figure 3.3: Preliminary Knowledge Layout

3.3 knowledge layout

By using prolog, we've restricted our inference capabilities to formal rule-based deduction. However, rule-based systems can be extended to allow reasoning beyond pure logic expressions, at least from a user's point of view. For problems where the domain can be expressed as a virtual model of the real world, so called model-based reasoning can be built on top of the rule set. The main advantage of model-based reasoning is that models are much more easy to grasp than abstract webs of interdependent rules. We have defined a preliminary model of the network domain and written RDF dictionaries for the elements in the model. Figure 3.3 shows the model in the popular Entity Relationship representation. Appendix B contains a listing of the RDF dictionaries.

The model in figure 3.3 defines `app`, `feature`, `host`, `proto`, `iface` and `model` entities. These represent applications, application features, networked hosts, protocols, network interfaces and network interface models, respectively. The relations between these entities, displayed as diamonds, show how they depend on one another. So, interfaces can connect to other interfaces, protocols implement application features (e.g. video streaming or encryption) and network interfaces are instances of a NIC model. The figure also shows the various entities and relationships belong to one of the colored areas. The single node area represents live information about a computer node and contains the node and local NIC entities. This information will mostly come from live resource sensors. The

reference database represents a dictionary of background information, such as what protocols exist and which services each offer. We encoded sample protocol and NIC model entries in static RDF dictionaries. Lastly, the task info area contains information about the running applications. This last area has not yet been encoded into RDF, but would incorporate application sensors.

The code snippet in figure 3.4 shows how the interface entity, its unspecified address and ipv4_address attributes and the connects relationship are defined in the RDF dictionary:

```
<rdfs:Class rdf:ID="interface" rdfs:comment="a network interface">
</rdfs:Class>

<rdf:Property rdf:ID="address">
<rdfs:comment>An identifier, in this context it applies to network interfaces</rdfs:comment>
<rdfs:domain rdf:resource="#interface"/>
</rdf:Property>

<rdf:Property rdf:ID="ipv4_address">
<rdfs:comment>a 32bit number, representing an IPv4 address</rdfs:comment>
<rdfs:subPropertyOf rdf:resource="#address"/>
</rdf:Property>

<rdf:Property rdf:ID="connects">
<rdfs:domain rdf:resource="#interface"/>
<rdfs:range rdf:resource="#interface"/>
<rdfs:subPropertyOf rdf:resource="file:/home/wdebruij/prolog_programs/v4-rdf_works/relations.rdf#bidir">
<rdfs:comment>the connects relationship can only be established between network interfaces</rdfs:comment>
</rdf:Property>
```

Figure 3.4: RDF networking dictionary codesnippet

Chapter 4

implementation

A prototype of a Network Inference Engine, based on the reasoning outlined in the previous chapters, has been built. We will here explain the individual modules and how they connect. In broad lines, the NIE implementation consists of 4 parts: the prolog inference engine, a basic interface for sensors input, a high-level networked interface for RDF data exchange and a human-computer interface, simplifying use of the RDF interface. We will discuss each of these in detail and give suggestion for connecting to the engine afterwards.

4.1 prolog-based inference engine

As a basis for the inference engine we use the SWI implementation of Prolog [Wie97], version 5.2.7, running on a Gentoo Linux machine with a Linux 2.6.0-test6 kernel. In general, any Prolog and OS combination would do, although the SWI implementation has a few perks: (1) it is extremely portable, (2) the sourcecode may be freely inspected and modified and (3) it comes standard with a large array of support packages. Especially of interest to us were the packages for C connectivity, HTTP data exchange and RDF parsing.

Since most of the knowledge is encoded in RDF triples, the internal rulebase of the Prolog interpreter can be quite small. In its most elemental form, the rulebase consists of solely of a general purpose pathfinding algorithm, used to walk the graph of RDF statements. However, for reasons of simplicity, often requested goals can be written as prolog rules directly. Figure 4.1 shows the (abbreviated) code for finding a route in the knowledge base. From the figure can be seen that all the routing rule does is concatenate a number of pathfinding goals. Since the RDF code is abstracted from the general rulebase, rules are written independent of the underlying syntax and are easily adapted to another encoding.

In the NIE implementation, the prolog interpreter is also used for other purposes than the inference alone. Most code actually deals with interfacing to the outside world. Figure 4.2 shows the mechanism SWI-Prolog supports for

```

% general purpose declaration of an edge
edge(Graph, From,To,Source):-
rdf(FromGlob, GraphGlob, ToGlob, Source).

% general purpose pathfinding algorithm
path(Graph,From,To, PathList):-
path_internal(Graph,From,To,[From], SubPathList),
PathList = [From | SubPathList].

% find a route between two hosts
route_host(From,To):-
path(net:'interfaces',From,FromIface, _),
path(net:'interfaces',To,ToIface, _),
path(net:'connects',FromIface,ToIface, OutPath).

```

Figure 4.1: abbreviated prolog ruleset for network routing

connecting to other applications. The NIE prolog source files are detailed in appendix B.

4.2 basic IO

To reason about networks, the inference engine needs to know certain basic facts, such as what devices exist, which hosts can be reached and the current status of these resources. To supply low-level information to the inference engine by manual addition of rules is not only a cumbersome task, but also largely useless, as the state of these low-level resources can change quite frequently. In section 3 we introduced the notion of sensors: dumb devices that output low-level information. Interfaces to a few sensors have been created, namely *procf*s and the linux kernel.

connecting to C Instead of writing individual interfaces for each sensor, we created a single interface between Prolog and C. Then, a special purpose C library was implemented to connect to the sensors. The first library, called `wjdb_prolog_c` is a wrapper around the standard SWI-Prolog C interface and contains general `prolog2c` and `c2prolog` routines, as well as a specific handler for our sensor interface library. Also, it contains an test application that demonstrates how the various interfaces work. When one wants to call the NIE from inside a C application, the `c2prolog` routines can be used to start, query and destroy the machine on the fly. The machine can be run from the current thread or start its own independent thread of execution.

C sensors interface The second interface, `wjdb_list`, contains individual handlers for the various kernelcalls and `proc` filereaders. While other interfaces can be added directly to prolog for new sensors, we urge developers to use the `wjdb_list` API whenever possible. The `wjdb_list` library contains most of the code needed for extracting information from external sources. Indeed, the reason this library was created was to increase development speed. Especially reading from linebased files, such as `/proc/net/dev` and `/proc/net/route` requires little extra

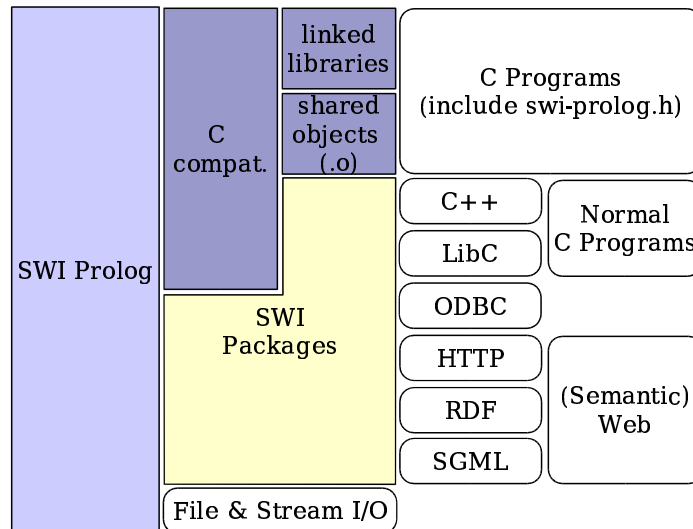


Figure 4.2: SWI-Prolog interfaces to the outside world

work per file.

At the moment, `wjdb_list` contains handlers for the previously mentioned files and for retrieving additional network interface information directly from the kernel through `ioctl(..)` requests. The handlers add their information to a linked list, which in turn is extracted and pushed into the knowledge base by the `wjdb_prolog_c` functions. A special set of prolog predicates then correlates this raw data, for instance by merging identical data from independent sources. This last step can be seen as an example of expert input, since the rules to transform the unstructured data into interconnected RDF triples stem from a higher understanding of the processed data.

generic prolog IO Prolog itself, finally, contains primitive IO constructs for reading and writing to files and calling predicates from the command-line. Scriptbased applications might prefer to call prolog directly in this low-level manner, instead of using the `c`-interface or the high-level interface.

4.3 distributed IO

The basic IO mechanisms are fine for reading foreign data and for connecting to the NIE locally, but another interface is needed to allow exchange of data between different NIEs and other possibly remote applications. For this purpose a higher-level distributed IO interface was built. Because we use RDF internally and SWI-Prolog contains much of the functionality for inputting and outputting

the RDF/XML serialization, addition of a high-level interface meant little work.

As a basis for exchanging information we use the widely used HTTP protocol. RDF is sent over raw HTTP, serialized as XML files. In the future, SOAP [W3C00] or another XML Remote Procedure Call layer might be preferred over raw HTTP, though.

The HTTP client and server functionality is part of the SWI-Prolog package library. The only alteration we made was to add RDF specific capabilities, e.g. serialization and deserialization routines. Also, by using the HTTP search field (the part of an URL after the question mark), it is possible to call prolog goals remotely. The server responds by sending an XML file containing the appropriate RDF data. Parsing of this data is left up to the caller application. A specific caller application one can use is a web browser. How we use web browsers for human computer interaction is described in the next section.

4.4 human computer interface

So far we've discussed techniques for connecting to the NIE from other programming environments. However, as both experts and end-users will often be humans, a graphical user interface can be considered a necessity. The combination of HTTP and XML we already use lends itself perfectly for web-enabled use. While many static websites are written in the HyperText Markup Language, a meaningful instantiation of XML similar to RDF/XML, modern browsers can display all XML encoded data. Since XML files lack style information, reading these in their raw form does not offer a significant advantage in itself, however.

To enable the display of XML data in a easy to understand format, we add style information in an additional file formatted in the *eXtensible Styleheet Language - transformation* (XSLt). XSLt, itself encoded in XML, contains instructions to translate XML data into other output types. One of the great uses of XSLt is to translate XML into human oriented HTML on the fly. Modern browsers ¹ natively support these stylesheets. All we need to do to add a graphical user interface to our existing system, therefore, is to append transformation information to the existing RDF/XML output. We would like to note that, since XSLt can transform any XML data, it can also be used to filter or search for information. Having built-in support for the XPath query language, many goals written in prolog can also be written in XSLt. At this time we see no reason to use this feature, however.

Without wanting to go into the matter of webpublishing to deeply, we do need to discuss some other techniques used for building the web frontend. The HTML code as outputted by XSLt only contains the layout of the output. Style information (e.g. font, letter size, colors), on the other hand, is included from yet another file: a Cascading StyleSheet (CSS). Keeping style information separated from the individual HTML files allows for quick alteration of the look-and-feel of a site and more importantly separates these concerns. The HTML output

¹in any case Internet Explorer 6SP1+ (can be flaky) and Mozilla 1.4+

1. functional, calling the NIE from C or Prolog code when needed
2. through the commandline
3. as a stand-alone entity, through HTTP

Figure 4.3: interaction options

also contains some programming code (Javascript) to modify data output at runtime.

In essence, the HCI adds a second processing layer to the output system. By using industry adopted languages (XML, XSLt, XHTML, CSS and DOM processing) it is possible to move this computation to the client, thereby leaving the inference engine lean. The HCI itself, too, is not too complex, but because of its reliance on an array of languages and the formal strictness of these languages, altering the interface can be difficult for untrained developers. Some introductory references to these technologies can be found in appendix A.

The graphical user interface can be reached at `\url{http://localhost:8080}` on a computer running the unmodified NIE.

4.5 interacting with the NIE

Summarizing, there are 3 ways to interact with the NIE, each shown in table 4.3. In 2.3, we asked the question where the NIE is to live. We were unable to answer that question satisfactory for large scale deployment. For demonstration purposes, however, the question is somewhat besides the point, and implementational issues will be more influential. We expect the functional interface to be the easiest to implement, the `wjdb_prolog` library contains example code that can be reused instantly. Most advanced is the HTTP environment, but using it means that clients must be able to parse XML (to some extent) and can connect to an HTTP server. Finally, the only use we see for the commandline interface is for calling the engine from scripting languages.

Chapter 5

related work

Applying a decision making architecture to network configuration problems is a highly specific domain. Therefore related projects are scarce. One recent project, however, is of particular interest. Sophia [WPR03], is envisioned to be a so called ‘information plane’, a distributed resource where data is published and knowledge can be obtained from. Our initiative closely resembles Sophia in many ways, but there are a few characteristic differences. First of all, Sophia is supposed to be a centralized knowledge base. That is, the application may be distributed, but the result is a central machine. The NIE, on the other hand, is user-centric. Furthermore, Sophia is built from the ground up, while we chose to use well known solutions wherever possible. Lastly, Sophia makes the case for keeping knowledge representation as basic as possible [RPKW03], while we propose the exact opposite.

The APNet initiative, whereof this enquiry is part, shares some of its research questions with the Knowledge Plane project [CPRW03]. For this project a similar, but more theoretical investigation has been carried out [QMM03]. They briefly mention RDF as a possible data exchange language, but shy away from making it central to the environment. Instead, they propose to use the Comming Information Model (CIM), defined by the Distributed Management TaskForce, because they believe it to have a large array of standardized semantics ranging from network devices (SNMP) to desktop management (DMI). The paper is brief on specifics, unfortunately. The only reason we can see for preferring CIM to RDF is the availability of datasources. However, apart from SNMP, deployed applications seem scarce. From an architectural point of view, the distributed approach of RDF should be preferred over the centralized, hierarchical data layout used in CIM, since it removes the need for governing bodies and increases language expressivity.

Combining logical inference with RDF has been proposed before [DBSA98, GLMB, Ogba, Ogbb, SD02]. The case for using Prolog as an inference engine with RDF as the knowledge base is made by Jan Wielemaker et al. [WSW03]. An actual application use Prolog and RDF, that unfortunately appears to have been abandoned, is the Mozillation inference engine for Mozilla [Bri]. To the

best of our knowledge so far no practical application using this combination of technologies exists. Model based reasoning in general, however, is being used frequently. A prime example of this technology is the reasoning engine used in NASA's Deep Space One autonomous probe: Livingstone [liv, MNPW98]

Chapter 6

current status and future work

We've created a prototype implementation of a network-oriented inference engine. Implementing an actual system gave rise to many questions, some of which we were able to answer. As such, we believe the main contribution of our work to be not the software, but the lessons learned from building it. That said, one may choose to continue on the work done so far by taking the NIE prototype and extending it. As a primer, we will now discuss some issues we feel must be resolved before an actual application may be considered useful.

The present implementation of our system contains a number small defects and functional shortcomings. Most prominent is the lack of actual prolog queries. The main reason for this is that no applications are using the system so far. Application developers will have to add prolog predicates to the knowledge base for their specific use-cases. While we tried to add useful sensors, one can expect to have to add these as well. The `wjdb_list` library was created to facilitate this process. One sensor we feel might be especially interesting is a tcp connection monitor (input from `/proc/net/tcp` or `netstat-t`). A Connection monitor can extract application preferences from their behavior. Future connections can then be rerouted transparently.

Taking a broader scope, the NIE lacks key features needed for adaptive networking: conflict resolution and service negotiation between actors in the network, a globalizing knowledge base overlay or remote NIE discovery mechanism, actor authentication, security mechanisms and knowledge quality control. Each of these fields justify in-depth enquiries in their own right. Especially conflict resolution and knowledge quality control are hotly debated topics. For the first, we suggest looking into solutions initially created for computational grids. Quality control might be carried out using a combination of stochastical reputation systems and more formally defined trust-management systems [BFL96], such as Keynote [BFIK99]. One tool worth exploring is Friend-Of-A-Friend (FOAF), an RDF dictionary for creating so called *webs of trust*. Because it uses

RDF, a FOAF based reputation system would fit perfectly in our knowledge base.

Chapter 7

concluding remarks

We introduced the Network Inference Engine, a prototype application for network-oriented decision making. When building such a system many questions crop up. Knowledge representation and the selection of an inference technology first and foremost, but we've also encountered many other issues. We've tried to justify our selection decisions and gave primers into available systems where we believe additional research is necessary.

For the representation of the knowledge base we selected the Resource Description Format, mostly for its decentralized nature, applicability to online resources and the widespread availability of supporting parsers and tools. For the inference engine we selected a rule-based system, namely Prolog. Rulebased systems are most simple in nature and therefore ideal for prototyping. However, we believe alternative technologies might be more suitable to this problem. Also, a hybrid solution, using prolog as the central parser and more advanced techniques as external experts, seems like a viable solution.

As an application, the NIE implementation we believe is still lacking, although mostly in content. The software is meant to be used for demonstrative purposes where security, robustness, stability and performance are less important than in real life. Much work needs to be done if it is to be moved out of the prototype stage. No plans currently exist, however, to advance the status of the software. For those interested, it is distributed under the OpenBSD license and may therefore be freely adapted.

Appendix A

further reading

We will now give a quick list of suggested readings. This is not a bibliography, nor a copy of the one at the end B.6, but a collection of links to online tutorials into the used and inspected technologies. Most of the resources are taken directly from my bookmarks and therefore freely available online.

A.1 decision making architectures

A.1.1 case-based reasoning

Case-Based Reasoning on the web

introduction to case-based reasoning

<http://www.cbr-web.org/CBR-Web/cbrintro/?info=index&menu=ai>

A.1.2 model-based reasoning

Skunkworks/Livingstone2 Documentation

information on the DS1 spaceprobe reasoning engine toolkit

<http://ic.arc.nasa.gov/projects/mba/projects/L2/doc/index.html>

here's more information about the research group behind Livingstone:

<http://ic.arc.nasa.gov/projects/mba/>

LPA Flex

overview of a commercial model-based reasoning environment called flex. Flex is particularly interesting, as it is built on top of prolog

http://www.sxst.it/lpa__flx.htm

LPA Flint

Flint adds support for reasoning under uncertainty to both Flex and Prolog

http://www.lpa.co.uk/ind_pro.htm

Free/Cheap Expert System Shells

part 5 of the *expert system FAQ*, this is an overview of expert system frameworks. The rest of the document might be interesting as well.

<http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/ai-repository/ai/html/faqs/ai/expert/part1/f>

A.1.3 prolog

Adventures in Prolog

an excellent introduction into Prolog

<http://www.amzi.com/AdventureInProlog/index.htm>

Building Expert Systems in Prolog

the cousin to the previous link. This site details what expert systems are and how you can implement one in prolog

<http://www.amzi.com/ExpertSystemsInProlog/xsipfrtop.htm>

duplicate link with a nicer layout :

<http://www.oopweb.com/Prolog/Documents/XSIP/VolumeFrames.html>

Guide to Prolog Programming

an example-driven prolog tutorial

<http://kti.ms.mff.cuni.cz/~bartak/prolog/contents.html>

Prolog and NLP

another course, this time from the University of Essex

<http://courses.essex.ac.uk/lg/LG519/1-Prolog/index.html>

Learn Prolog Now!

yet another prolog tutorial, adds more detail on how prolog works

<http://www.coli.uni-sb.de/~kris/prolog-course/html/index.html>

SWI-Prolog reference manual

The manual of the interpreter we used

<http://www.swi.psy.uva.nl/projects/SWI-Prolog/Manual/>

Prolog Syntax reference

a detailed discussion of the prolog syntax, from the Sicstus Prolog site

http://www.cs.bham.ac.uk/~pjh/prolog_course/sicstus_manual_v3_5/sicstus_42.html

CPU Prolog Repository

the prolog subtree of the CMU AI Repository, contains information and code

<http://www-2.cs.cmu.edu/afs/cs/project/ai-repository/ai/lang/prolog/0.html>

SWI-Prolog HTTP support

information about the HTTP support package for SWI-Prolog

<http://gollem.swi.psy.uva.nl/cgi-bin/pl-cvsweb/pl/packages/http/http.html?rev=1.2>

A.2 knowledge representation

PlanetLab Sensor Server

an implementation of the Sophia ?? information plane

<http://www.cs.princeton.edu/~scott/PlanetLab/>

Open Directory Project KR subtree

links to many resources, including RDF and the Semantic Web

http://directory.google.com/Top/Reference/Knowledge_Management/Knowledge_Representation/?il

A.3 XML and XSLt, XHTML and CSS

W3C XML home

contains all specifications of XML

<http://www.w3.org/XML/>

XML.com

has nice articles on all things XML

<http://www.xml.com/>

W3C XSLt Home

contains all specifications of XSLt

<http://www.w3.org/TR/xslt>

What's XSLt

an introductory overview of XSLt and XPath at XML.com

<http://www.xml.com/pub/a/2000/08/holman/>

HTMLHelp.com

contains complete reference listings of HTML and CSS syntax, an unskippable resource for writing standards abiding websites

<http://www.htmlhelp.com>

W3C Markup Validation Service check your XML, HTML and CSS files for correctness at the W3C validator (seriously, do it!)

<http://validator.w3.org/>

A.4 RDF

W3C RDF homepage

the main RDF resource. Contains specifications for everything related to RDF. Also of interest might be the XML and XSLt specification sites
<http://www.w3.org/RDF/>

Dave Becketts' RDF Resource Guide

the yahoo for RDF
<http://www.ilrt.bristol.ac.uk/discovery/rdf/resources/>

RDF Schema document

introduces two 'standard' RDF dictionaries: rdf and rdfs
http://www.w3.org/TR/rdf-schema/#ch_bag

Dublin Core Metadata Initiative

a much used RDF dictionary. The DC dictionary defines general metadata tags, such as 'creator' and 'lastmodified'
<http://dublincore.org/>

Web Ontology Language: OWL

paper on OWL, RDF's latest and greatest cousin
<http://www.cs.vu.nl/~frankh/abstracts/OntoHandbook03OWL.html>

4Suite

some open source RDF parser. I haven't tried this, though
<http://4suite.org/index.xhtml>

A.4.1 Tutorials

RDF Primer

W3C's introductory document on RDF
<http://www.w3.org/TR/2003/WD-rdf-primer-20030905/>

RDF Made Easy

a nice introduction, as close to RDF for dummies as it gets
<http://home.ccil.org/~cowan/XML/RDF-made-easy.html>

RDF Hints and Tips

priceless
<http://infomesh.net/2001/08/swtips/>

Learning RDF through N3

Notation3 is an alternative syntax for RDF and easier to learn than XML/RDF

<http://www.w3.org/2000/10/swap/Primer>

A.4.2 Non Prolog RDF Inference

RDF Inference Language (RIL)

one of the many proped inference languages for RDF

<http://xml.coverpages.org/RIL-20010510.html>

Thinking XML: RDF Query using Versa

an IBM developerworks article about RDF inference, this time using Versa

<http://www-106.ibm.com/developerworks/xml/library/x-think10/>

A.4.3 Prolog and RDF

An introduction to Prolog and RDF

very good, be sure to read the second article (below) as well

<http://www.xml.com/pub/a/2001/04/25/prologrdf/index.html>

RDF Applications with Prolog

a cousin to the previous link. Explores a real use-case for Prolog+RDF

<http://www.xml.com/pub/a/2001/07/25/prologrdf.html>

the SWI-Prolog RDF parser

information on one of the RDF tools in SWI-Prolog

<http://www.swi-prolog.org/packages/rdf2pl.html>

Online SWI-Prolog RDF parser demo

try out an inference engine from the comfort of your own webbrowser

<http://www.swi.psy.uva.nl/projects/SWI-Prolog/packages/sgml/online.html>

SWI-Prolog/XPCE Semantic Web Library

the manual of the most important RDF packages for SWI-Prolog

<http://www.swi-prolog.org/packages/semweb.html>

Mozilla RDF / Enabling Inference

the website of the Mozillation Prolog-based inference engine plugin for RDF.

Hasn't been edited in a long time

<http://www.mozilla.org/rdf/doc/inference.html>

A.4.4 Semantic Web

The Semantic Web

A Scientific American article introducing the vision of a semantic web to a broad

audience. Written by Tim Berners-Lee.
<http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>

The Sematic Web: It's whom you know
a critical view of the semantic web and its future, food for cynics.
<http://www.oreillynet.com/pub/a/network/2002/04/19/platform.html>

W3c Semantic Web homepage
all things semweb
<http://www.w3.org/2001/sw/>

Jena 2
a semantic web framework built on top of Java
<http://www.hpl.hp.com/semweb/jena2.htm>

A.4.5 tools

IsaViz
probably the best RDF visualization tool out there. Can also be used as an editor. Caution: performance is horrible ('cause it's java-based)
<http://www.w3.org/2001/11/IsaViz/>

RDFAuthor
a nice graphical RDF editor. Less sophisticated than IsaViz
<http://rdfweb.org/people/damian/RDFAuthor/Tutorial/Tutorial1/>

SMORE
a text-based RDF triple editor
<http://www.mindswap.org/~aditkal/editor2.shtml>

A.5 other

A.5.1 trust management

KeyNote
homepage of the KeyNote trust management framework
<http://www.cis.upenn.edu/~keynote/>

SDSI
website of the SDSI 2.0 toolkit for trust management
http://theory.lcs.mit.edu/~cis/sdsi/sdsi2/sdsi20_1.html#SEC1

Finding friends with XML and RDF

an IBM developerworks article on FOAF

<http://www-106.ibm.com/developerworks/xml/library/x-foaf.html>

TrustFlow

overview of a *distributed* reputation system for blogs. Not too practical, but might give you some ideas on where to proceed

<http://www.livejournal.com/community/trustmetrics/2363.html>

SmartMobs

article discussing distributed trust metrics

<http://www.smartmobs.com/archives/001430.html>

references this slashdot discussion:

<http://ask.slashdot.org/article.pl?sid=03/08/07/233226>

A.5.2 distributed messaging

Using RDF with SOAP

an IBM developerworks article. SOAP is the most well-known remote procedure call mechanism using XML. It can also be used as a simple XML-aware transport layer over HTTP

<http://www-106.ibm.com/developerworks/webservices/library/ws-soaprdf/>

Jabber

introduction to a general purpose overlay network called Jabber. Could be useful as a basis for a globalized knowledge-base

<http://www.jabber.org/about/techover.html?PHPSESSID=873bae02b7abfc3e55558b5288bf5817>

A.5.3 APNets

APNet project homepage intro and papers

<http://www.cis.upenn.edu/~dsl/APNet/>

Appendix B

software listing

The NIE software package contains C libraries, RDF files, prolog sourcecode and webfiles. Below you can find a short listing of what each package and file does. Additional comments can be found inside the individual files.

B.1 prolog sourcecode

the prolog sourcecode can be found in `emph/prolog`. It contains the bulk of the code, including APNet specific rules, the HTTP networking interface and RDF predicates. The directory contains the following files :

name	contents
<code>config.pl</code>	configuration options encoded as prolog predicates
<code>main.pl</code>	the initialization file, called from <code>run.sh</code>
<code>rdf_httpd.pl</code>	everything having to do with the webserver
<code>rdf_import.pl</code>	predicates for retrieving RDF data
<code>rdf_init.pl</code>	predicates for declaring RDF namespaces and optionally importing static information
<code>rules.pl</code>	contains the RDF to graph translation rules and some sample APNet specific predicates
<code>run.sh</code>	starts up the inference engine, the RDF knowledge base and the webserver
<code>wjdb_list.pl</code>	calls the <code>wjdb_list</code> C routines and copies live system information to the RDF database. E

B.2 wjdb_prolog_c library

The `wjdb_prolog_c` library, located in `/libwjdb_prolog_c`, contains connection routines for calling prolog from C and calling C routines from prolog. Much of its functionality is no more than a wrapper around SWI-Prolog's C interface. However, we created a special interface for the `wjdb_list` library discussed below. Therefore this library depends on the `wjdb_list` library and must be built after that. Install this library in `/usr/local` by issuing a standard `./configure&& make && make install`.

name	contents
c_to_prolog_wjdblist.*	allow prolog predicates to reference wjdb_list code
c_to_prolog.*	general registration routines needed for SWI Prolog's C interface
Changelog	overview of alterations
configure	needed for building the binary package (standard automake tool)
README	short helpfile
prolog_to_c.*	C routines for calling prolog predicates
test_wjdb_prolog_c.*	example executable, showing what the library can do

Other files in the directory are part of the automake process and unimportant.

B.3 wjdb_list library

The wjdb_list library (in /libwjdb_list) contains handlers for reading input and writing it to a standardized structure. It takes care of most functionality, such as reading from linebased files (e.g. /proc/net/dev) for you. The individual handlers use this functionality for their respective sensor sources. Install this library in /usr/local by issuing a standard ./configure&& make && make install.

name	contents
Changelog	overview of alterations
configure	needed for building the binary package (standard automake tool)
README	short helpfile
test_wjdb_list.c	sample executable using the package
wjdb_list.h	main header file
wjdb_list_handler.*	reusable functionality
wjdb_list_handler_registration	needed for registering new sensors
wjdb_list_ioctl_dev.*	sensor reader for linux kernel ioctl(..) calls
wjdb_list_proc_net_dev.*	reads input from /proc/net/dev
wjdb_list_proc_net_route.*	reads input from /proc/net/route
wjdb_list_unknown.*	fallback routines for if anything goes wrong

Other files in the directory are part of the automake process and unimportant.

B.4 rdf dictionaries

Directory /rdfschemata contains a few RDF dictionaries. Please note that these can contain errors, as I had to learn RDF while writing them.

name	contents
comp.rdf	information about computers
iface_models.rdf	NIC model example database
networking.rdf	information about networking devices and routes
protocols.rdf	information about protocols
relations.rdf	information about abstract relationships
sampe.rdf	a sample set of resource instantiations, might be a bit outdated compared to actual pr
services.rdf	information about services

B.5 human computer interface

In the directory /www you can find the static components that make up the HCI :

name	contents
index.html	simple page containing links to the rest of the site
standard.xsl	XSLt transformation page for generic RDF information
standard.css	CSS stylesheet that adds some eye candy to our otherwise bland output
prolog.html	Prolog interface: Execute queries from here. Contains ready-made example queries
doc/	Subdirecory that contains this documentation parsed as html
src/	Subdirecory with HTML snippets that make up the site after running 'make'

B.6 other

directory /doc has a /src subdirectory where you can find the latex sourcecode for this manual, together with some conversion tools. Running make in this directory will output postscript, pdf and html versions of the documentation.

Bibliography

- [apn] Apnet public website. Available from World Wide Web: <http://www.cis.upenn.edu/~dsl/APNet>.
- [AvH03] Grigoris Antoniou and Frank van Harmelen. Web ontology language: Owl. In S. Staab and R. Studer, editors, *Handbook on Ontologies in Information Systems*. Springer-Verlag, 2003.
- [BFIK99] Matt Blaze, Joan Feigenbaum, John Ioannidis, and Angelos D. Keromytis. The KeyNote trust management system. Internet Request for Comment RFC 2704, Internet Engineering Task Force, September 1999. Available from World Wide Web: <http://www.ietf.org/rfc/rfc2704.txt>. Version 2.
- [BFL96] Matt Blaze, Joan Feigenbaum, and Jack Lacy. Decentralized trust management. Technical Report 96-17, AT&T Research, 28, 1996. Available from World Wide Web: citeseer.nj.nec.com/blaze96decentralized.html.
- [BLHL01] Tim Berners-Lee, James Hendler, and Ora Lasila. The semantic web. *Scientific American*, May 2001. Available from World Wide Web: <http://www.sciam.com/article.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21>.
- [Bri] Daniel Brickley. Mozilla rdf / enabling inference. Available from World Wide Web: <http://www.mozilla.org/rdf/doc/inference.html>.
- [CPRW03] David D. Clark, Craig Partridge, J. Christopher Ramming, and John T. Wroclawski. A knowledge plane for the internet. In *Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 3–10. ACM Press, 2003.
- [DBSA98] S. Decker, D. Brickley, J. Saarela, and J. Angele. A query and inference service for rdf, 1998. Available from World Wide Web: <http://www.w3.org/TandS/QL/QL98/pp/queryservice.html>.

- [GLMB] R.V. Guha, Ora Lassila, Eric Miller, and Dan Brickley. Enabling inference. Available from World Wide Web: <http://purl.org/net/rdf/papers/QL98-enabling>.
- [Hop00] Adrian A. Hopgood. *Intelligent Systems for Engineers and Scientists*. CRC Press, 2000.
- [liv] Skunkworks/livingstone2 documentation. Available from World Wide Web: <http://ic.arc.nasa.gov/projects/mba/projects/L2/doc/index.html>.
- [MNPW98] Nicola Muscettola, P. Pandurang Nayak, Barney Pell, and Brian C. Williams. Remote agent: To boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2):5-47, 1998. Available from World Wide Web: citeseer.nj.nec.com/article/muscettola98remote.html.
- [Ogba] Uche Ogbuji. Ril: A taste of knowledge. Available from World Wide Web: <http://www.xml.com/pub/a/2000/10/11/rdf/ril.html>.
- [Ogbb] Uche Ogbuji. Thinking xml: Basic xml and rdf techniques for knowledge management, part 6: Versa. Available from World Wide Web: <http://www-106.ibm.com/developerworks/xml/library/x-think12.html>.
- [QMM03] Stephen Quiroigico, Kevin Mills, and Doug Montgomery. Deriving knowledge for the knowledge plane, 06 2003. Available from World Wide Web: <http://www.isi.edu/know-plane/DOCS/steveq.ontologies.pdf>.
- [RPKW03] Timothy Roscoe, Larry Peterson, Scott Karlin, and Mike Wawrzoniak. A simple common sensor interface for planetlab. Technical report, planetlab.org, 05 2003. Available from World Wide Web: <http://www.planet-lab.org/pdn/pdn03-010.pdf>.
- [SD02] Michael Sintek and Stefan Decker. Triple-a query, inference, and transformation language for the semantic web. In *Lecture Notes in Computer Science*, volume 2342/2002, January 2002. Available from World Wide Web: <http://www.springerlink.com/openurl.asp?genre=article&iissn=0302-9743&volume=2342&>
- [W3C00] W3C. *Simple Object Access Protocol (SOAP) 1.1*, 2000. URL: <http://www.w3c.org/TR/SOAP>.
- [Wie97] J. Wielemaker. Swi-prolog reference manual, 1997. Available from World Wide Web: citeseer.nj.nec.com/wielemaker96swiprolog.html.

- [WPR03] Mike Wawrzoniak, Larry Peterson, and Timothy Roscoe. Sophia: An information plane for networked systems. Technical report, planetlab.org, 07 2003. Available from World Wide Web: <http://www.planet-lab.org/pdn/pdn03-014.pdf>.
- [WSW03] Jan Wielemaker, Guus Schreiber, and Bob Wielinga. Prolog-based infrastructure for rdf: Scalability and performance. In *Proceedings of the 2nd International Semantic Web Conference 2003*, volume 2870/2003, pages 644–658, 2003. Available from World Wide Web: <http://www.springerlink.com/openurl.asp?genre=article&issn=0302-9743&volume=2870&>