Logic Programming Languages for the Internet

Andrew Davison

Prince of Songkla University
Dept. of Computer Engineering
Hat Yai, Songkhla 90112, Thailand
dandrew@ratree.psu.ac.th

Abstract. We specify the major characteristics of the Internet under the headings: heterogeneity, service characteristics, dynamic nature, no global notions, and unreliability (i.e. security and partial failure). In the process, we identify five categories of Internet services: hosts, active entities, agents, semistructured data, and passive code.
Logic Programming (LP) languages for the Internet are divided into six broad groups: shared variables, coordination, message passing, client-side execution, server-side execution, and integration of heterogeneous data sources. Within each group we attempt to highlight the advantages and disadvantages for Internet programming in terms of our Internet characteristics and services, and describe LP languages that typify the group.

1 Answering the Challenge

In the mid 1980's, Carl Hewitt argued that Logic Programming (LP) was inadequate for modeling open systems [67]. Hewitt's objections rest on classical logic's use of a static, globally consistent system which cannot represent dynamic activities, inconsistencies, and non-global concerns.

At the time, his broadside was addressed by two papers. Kowalski [77] agreed that model theoretic formulations of logic were lacking, and proposed the use of knowledge assimilation to capture change, along with additional elements to deal with belief systems. Kowalski's subsequent work on the event calculus and reactive and rational agents [78, 79] can be viewed as developments of these ideas. Kahn and Miller suggested concurrent LP as the logical framework for open systems [75].

Another way of answering Hewitt is to look to the Internet, the World's largest open system. Among other things, this survey shows that LP is a successful *component* of Internet programming languages, employed for tasks ranging from security semantics, composition of heterogeneous data, to coordination 'glue'. Many approaches have moved beyond first order logic (e.g. to concurrent constraints, linear logic, higher order), and LP is frequently combined with other paradigms (e.g. mutable state, objects). Furthermore, many of the programming concerns for the Internet are still less than fully understood: for example, there are unanswered problems related to mobility, security, and failure. No single Internet language, including an Internet LP language, has all the answers *yet*.

2 From LAN to Internet

Cardelli [25] argues that the Internet is not some scaled-up version of a LAN, but actually violates our familiar assumptions about distributed systems, learnt from multiprocessor and LAN-based applications. We specify the differences under five headings: 1) Heterogeneity; 2) Service Characteristics; 3) Dynamic Nature; 4) No Global Notions; and 5) Unreliability.

2.1 Heterogeneity

Due to wide variations in Internet components, it is impossible to make predictions about overall behaviour. This covers topics such as processor capabilities, available resources, response time, and latency. In particular, bandwidth fluctuations due to unpredictable congestion or partitioning means that any guarantees of services will be guesses at best, and that failure becomes indistinguishable from long delays.

In the next sub-section, we identify five kinds of Internet service: hosts, active entities, agents, semistructured data, and passive code. Each of these exhibit quite different capabilities and behaviours, and have a variety of owners and authors.

2.2 Service Characteristics

Hosts Hosts are services which stay in one (virtual) location, perhaps acting as a Web server, virtual machine, or resource provider. Virtual locations are defined in terms of their position within administrative domains, through which mobile services (e.g. agents) must move. Domains may be nested, and a mobile service must have suitable capabilities to move in and out of those domains it visits.

The utilisation of administrative domains (a non-technical constraint on programming) gives rise to a view of the Internet as a hierarchical set of spaces with non-flat addressing, non-transparent routing, and non-free mobility – a radical departure from a LAN-derived model [26].

Active entities Typically criteria for an active entity are autonomy, reactivity, concurrency, and non-determinism [67, 75].

The arguments for employing the object oriented paradigm as part of an Internet model are so strong that other paradigms, such as LP, must find ways to reconcile themselves with it. Several recent object oriented deductive languages are compared in [82]. An older survey of LP-based object oriented languages is [44].

The disconnected nature of entity communication is often modeled by asynchronous message passing (to decouple communication delay from computation). Other reasons for an arm's length relationship are to maintain security and prevent failure of one entity affecting others.

Active entities may be mobile, deciding for themselves where to go – this is part of their autonomous behaviour. The inclusion of mobility into a programming language and/or system makes network transparency very hard to maintain [19]. Mobile entities require knowledge about current node (or host) connectivity and the location of site-specific resources. Mobility can mean different things: moving the entity's code, moving a reference to the entity (its execution location remaining unchanged), the entity's state, or relocating its computation or closure. Designing suitable abstractions for these approaches, especially inside an open system architecture, are difficult problems.

Agents There are various definitions of agents [60, 55], which are considered in more depth elsewhere in this volume. We will not discuss agent theories or languages.

Semistructured Data Semistructured data on the Web/Internet has been investigated extensively by Abiteboul [1]. It has an irregular structure (usually involving nested or cyclic structures), a mix of typed and untyped elements, and parts of the data are implicitly structured or partially specified. Significantly, these kinds of structural requirements are beyond the representational powers of many calculi and algebras.

Two important aspects of semistructured data on the Web is how to query it, and how to combine data from heterogeneous sources.

Passive Code A familiar example of passive code is the Java applet. In terms of functionality, passive code is similar to an active entity, the main difference being its means of mobility. An active entity decides for itself where to go, while another service (e.g. a Web browser) decides whether to download passive code. It is in that sense that the code is passive – it does nothing until another service executes it. Passive data mobility (sometimes called code fetching) is usually implemented via copying, while active entities actually move. This has consequences for the semantics of updates.

2.3 Dynamic Nature

Nothing is permanent on the Internet: all the services described in section 2.2 can change over time, new services appear, others disappear or move. This implies the need for white and yellow page services but, due to the lack of global knowledge, they will never be able to index everything. In any case, administrative domains will hide certain services.

Connectivity will also change, perhaps because of hardware issues, or because logical link information is passed around the system. Assumptions based on topology and routing behaviour will be suspect – for example, that messages arrive in the order they are sent.

2.4 No Global Notions

The Internet has no global time or state (knowledge base). Administrative domains mean there is no single naming scheme. Control/coordination is decentralised: to avoid bottlenecks in communication, because it is more scalable, and due to the impossibility of organising a single locus of control or coordination.

2.5 Unreliability

Security There can be no global trust, which implies the need for security. Until recently, security issues were concerned with the safe handling of messages, including the problems of eavesdropping, masquerading, tampering, and replay [39]. With the growing importance of mobile code, security concerns have become more complex.

Moore [95] identifies two problematic areas of mobility: a host requires protection against malicious mobile services (i.e. active entities, agents, downloaded passive code), and a mobile service must guard itself against dangerous hosts.

Thorn [123] considers four issues in the malicious mobile service category: maintaining the confidentiality of a host's private information, retaining the integrity of its information, preventing denial of service attacks, and visitor authentication. These issues are complicated by security being potentially 'spread' through many layers in a host.

Security is arguably at the heart of mobile entity design, since it defines the boundaries of control and responsibility between the entity and its host's execution environment.

Partial Failure Partial failures of the Internet fall into two main parts: node (host) failure and network link failures, with numerous subdivisions. For example, Coulouris et al. [39] identify node failures where the current state is lost, corrupted, or an earlier state is restored on restart. Other types of error, which may be caused by node or link failure, include message loss, message corruption, or message delay beyond some time limit.

In practice, fault tolerant protocols utilise time-outs and retransmission of messages based on assumptions about the maximum likely response time of a service and the likelihood of repeated message loss. Another approach is to use transactions, either implicitly added to the client by the system, or as a language construct. A popular language solution is to pass detected failures to the client as exceptions.

Waldo et al. [128] names partial failure as one of the four areas which makes distributed computing fundamentally different from local computation (the others are latency, memory accesses, and concurrency). Waldo believes that it is impossible to satisfactorily integrate failure handling for local and remote entities without introducing unacceptable compromises.

3 Let the Games Commence

This survey utilises the classification structure given in Figure 1. Within each category, we identify the advantages and disadvantages for Internet programming, and describe languages which typify that category.

- Shared Variables
  - Concurrent (Constraint) LP
  - Distributed Oz/Mozart
- Coordination
  - Linda and LP
  - Parallel Multiset Rewriting
- Message Passing
  - IC-Prolog II and Friends
  - PVM-Prolog
  - Mercury
- Client-side Execution
  - Libraries
  - Query Languages
  - Web Pages as Programs
  - Java and LP
- Server-side Execution
  - Libraries
  - Modifications to CGI
- Integration of Heterogeneous Data Sources
  - Classical Integration
  - New Wave Integration

Fig. 1. Categories of LP Languages for the Internet.

A useful online resource for Internet and Web programming using LP and constraint LP is `http://www.clip.dia.fi.upm.es/lpnet/index.html`. It contains links to numerous workshops and other resources.

4 Shared Variables

The languages described here utilise shared logic variables as a communications mechanism between processes. This approach raises many questions: Are shared variables alone expressive enough for the range of communication protocols required? Can such a technique be efficiently implemented? Are shared variables really any better than conventional communications devices (e.g. message passing, mailboxes)? What do the introduction of these ideas do to the semantics of the logic language?

Most research has addressed the communications aspects of using shared variables. Little consideration has been given to partial failure, security or mobility.

4.1 Concurrent (Constraint) LP

We will not discuss the concurrent LP and concurrent constraint LP paradigms in depth, as they are considered elsewhere in this volume; we also recommend [109, 122, 105]. The impact of concurrent LP on client-side and server-side Web programming is deferred to sections 7 and 8.

Paradigm Features Concurrent LP offers a process reading of logic programs, which is quite different from sequential LP, but well suited for the implementation of reactive, dynamically changing systems with encapsulated mutable state.

There are no global notions in concurrent LP languages: communication is based solely on shared variables which must be explicitly allocated between the processes. A process network is equivalent to a conjunction of AND-parallel goals.

Programs utilise very fine grained parallelism at the level of individual goal reductions. Even in distributed memory multiprocessor implementations, the overhead of goal management (e.g. task creation, migration to a processor, switching, scheduling, communication) compared to the amount of computation in the goal is of concern [5]. Across the Internet, it becomes imperative that task granularity be controlled. One solution is to add extra notation to the language for grouping related predicates and goals.

Programming Techniques The logic variable is an expressive communications mechanism; for instance, it can be used to encode stream communication (sequenced message passing), channels (partially ordered message streams), point-to-point delivery, multicasting, many-to-one links (stream merging), and blackboards.

Logix is a high-level environment/OS for developing FCP programs, written entirely in FCP [68]. Logix offers four layers of control over programs, defined using meta-interpreters. The failure mechanism only catches logical process errors not hardware/node failure.

Meta-interpreters are also used for implementing process to processor mapping [121]. Again a layered approach is utilised, with the program being mapped to a virtual machine topology, and then combined with a separate layer mapping the virtual machine to the physical architecture. At the program level, goals are positioned dynamically on virtual nodes using LOGO-like turtle annotations.

Process to processor mapping is a powerful abstraction away from the physical network. However, its success relies on the closely linked behaviour of the underlying architecture. The less well-ordered topology of the Internet may prove harder to abstract.

Efficient Communication A severe concern for language designers is the cost of output unification. In the case of FCP(:) the tell part of the guard carries out general unification, but this must be undo-able if the clause is ultimately not selected [76]. The cost of such atomic unification is similar to an atomic transaction, only possibly more prohibitive since partial bindings of data structures may be involved.

Strand makes the assumption that there is only one producer for a variable, thereby avoiding the "multiple tellers" problem altogether [58]. Janus is more restrictive in that there must only be one consumer and producer of a binding [106]. Strand and Janus programs cannot fail due to conflict bindings, and so do not have to implement atomic unification. Strand goes further and simplifies output binding to be assignment only.

The designs used in Janus and Strand have major consequences for Internet-based concurrent LP: they show that efficient communication between processes using logic variables is possible (i.e. by using non-atomic publication) while retaining their expressiveness.

PCN (Program Composition Notation) has been described as a hybrid of Strand and C, based around a few concepts: concurrent composition, single assignment variables, and nondeterministic choice [59]. Mutable variables are included, mainly as an optimisation, and to interface to pre-existing code. PCN utilises the mapping mechanisms, scheduling pragma, and virtual and physical topologies found in Strand.

PCN is novel in showing that the single assignment variable can be used as communications 'glue' for code written in non-logic programming languages.

Another approach to the multiple teller problem is to include special-purpose many-to-one communication mechanisms in the languages.

DRL [53] introduces logic channels as an efficient version of shared logic variables. Logic channels can contain messages made up of terms, logic variables, or other logic channels, thereby allowing dynamic reconfiguration of communication paths. This mechanism proved so useful that it was later extracted from DRL to become a coordination model suitable for any type of language [52]. The model also supports a virtual machine layer for grouping processes. PVM is used to implement remote process creation and message passing.

Failure Concurrent LP languages handle logical failure in one of two ways: those from the Concurrent Prolog 'camp' utilise meta-interpreters as described above for Logix. The other method, typified by Parlog, is to use a meta-call primitive [35].

The most developed concurrent LP language for failure handling is Sandra [54]. It borrows the guardian mechanism from Argus as a way of encapsulating processes, mutable state, a communications interface and a many-to-one communication port. A guardian instance represents a fail-stop logical multiprocessor. Every guardian has stable persistent storage (used in the recovery protocol), and can be relocated to a working processor in the event of failure.

Sandra utilises both forward and backward recovery. Forward recovery is based on exceptions (both logical and at the node level), which are processed by a separate handler in each guardian. Its semantics are specified using a meta-interpreter. Backward recovery is based on optimistic recovery using periodic logging and checkpointing.

The following predicate can signal (raise) an exception when a `get` message cannot be satisfied. `no-value` exceptions are dealt with by code in `resource/2` itself, as are hardware connection errors. Unprocessed exceptions will be passed out to the enclosing handler.

```
resource([get(X)|In], [X|L]) :-    % normal behaviour
  resource(In, L).
resource([get(X)|In], []) :-       % raise exception
  signal( no-value(get(X)) ).

resource(In, L) :-                 % handle exceptions
  otherwise( no-value(get(X)) ) |
  X = nothing, resource(In, L).
resource([_|In], L) :-
  otherwise( connect-error(resource) ),
  emergency_action(resource, In, L).
```

Security Security has been poorly considered by concurrent LP languages, perhaps because of the prevalence of uniprocessor and LAN-based implementations. Shapiro [109] reports that one use for read-only variables in Concurrent Prolog is to protect process communication across trust boundaries. The essential idea is to make the incomplete part of the output data structure read-only to its consumers, and keep write access inside the issuing process.

A related approach, implemented in FCP so that read-only variables could be used, is described in [94]. A protocol similar in style to public key encryption is proposed which uses pairs of unforgeable IDs issued by a dedicated 'locksmith' process.

4.2 Distributed Oz

Distributed Oz (or Oz 3) is a rich mix of paradigms and techniques, including symbolic computation, inference, objects, concurrency, and distribution [66].

Oz 3 introduces distributed programming and fault detection (Mozart is the name of the current implementation [125]). The main aim is to make programs as network transparent as possible – a program should perform the same computation independently of how it is partitioned over the network. In other words, Oz 3 tries to support the illusion of a single network-wide abstract store / address space, making distributed execution indistinguishable from concurrent evaluation. For example, no program level distinction is made between local and remote references.

Distributed Semantics The intention is to make the distributed semantics a natural extension of the centralised (local) meaning of a program, the practical benefit being that a stand-alone program requires virtually no changes to convert it to a distributed application.

Centralised and distributed semantics are presented in [65], and state (mutable pointers) is considered in [126, 124]. The main addition to the distributed semantics is site location for program entities.

Logic Variables Oz shows that logic variables can be efficiently supported in a distributed programming language while retaining the expressiveness seen in concurrent (constraint) LP [65]. In addition, Oz can make use of other aspects of the language (e.g. state, search) to improve on these algorithms. An example is the use of ports for many-to-one communication, which have constant time behaviour compared to the cost of concurrent LP merge networks (at best O(log n) for n senders).

Explicit networking benefits of logic variables include increased latency tolerance since producers and consumers are decoupled, and third-party independencies. This arises when two variables have been unified at a site, since that site is no longer involved in the network communication of future bindings.

Logic variables have proved so useful that the Oz group has added them to versions of Java (called CCJava) and ML [65]. Their main use in CCJava is as a replacement for monitors, allowing Java to utilise the dataflow threads techniques of Oz [112]. This requires the addition of statement-level thread creation.

Their approach is very similar to the extensions to C++ in Compositional C++ (CC++) [27], which include single assignment variables (called sync variables) and statement level threads. When a read is attempted on an unbound sync variable it causes the thread to block. CC++ also includes atomic functions, similar to synchronized methods in Java.

Beyond Network Transparency Although network transparency is a commendable design goal for distributing concurrent programs over a network, it seems unlikely to be wholly successful for Internet-based applications, such as mobile agents. However, Oz does support these kinds of programs by means of tickets and functors.

A ticket is a global reference (usually a URL) to an Oz entity (usually a logic variable), stored as an ASCII string. A ticket can be accessed by a system newcomer to obtain a language level communication link.

Functors are module specifications that list the resources that a module needs in order to execute. Their purpose is tied to an Oz programming technique called remote compute servers, which are procedures executed at fixed sites.

This example shows producer and consumer threads communicating via a data stream, with the consumer located remotely on a compute server:

```
proc {Generate N Max L}       % outputs N to Max-1 integers
  if N < Max then L1 in
```

```
    L=N|L1  {generate N+1 Max L1}
  else L=nil end
end

fun {Sum L A}             % return A + sum of L's elements
  case L
  of nil then A
  [] X|Ls then {sum Ls A+X}
end

local CS L S in
  CS={NewComputeServer 'sinuhe.sics.se'}  % remote server
  thread L = {Generate 0 150000} end      % local producer
  {CS proc {$} S={Sum L 0} end}           % remote consumer
  {Print S}                             % print result locally
end
```

Fault Tolerance Oz 3's support for fault tolerance is based on the assumptions that sites are fail-stop nodes (i.e. that permanent site failure is detectable) and that network failure is temporary. The basic response is to raise an exception when failure is detected, which can either be handled automatically or be processed by a user-defined procedure call. Typical default behaviour for network failure is to attempt to restart the TCP connection using a cache of existing TCP connections.

Van Roy [124] has shown how the centralised and distributed semantics for mutable pointers can be extended to include exception raising when a fault occurs.

The search for higher-level abstractions for fault tolerance in Oz is an active research goal. One approach is the *global store*, a globally fault-tolerant transactional memory, implemented as a Mozart library [6]. Internally, the store uses process redundancy: with n copies of a process it can tolerate up to *n-1* fail-stop process failures. There is an agent API on top of the store which provides fault tolerance and agent mobility without site dependencies.

Security Security concerns are still under investigation in Oz. Language security centers around lexical scoping, first-class procedures and the unforgeability of variable references.

An interesting point is the appearance of read-only views of variables (called futures). As in Concurrent Prolog, they allow the scope of variables to be limited so that, for instance, a stream cannot be altered by its readers [89]. Unlike Concurrent Prolog, the read-only mechanism imposes no efficiency penalty when it is absent.

Below the language level, Oz uses a byte-code emulator to protect machine resources, and can create virtual sites. A virtual site appears as a normal site, but its resources are under the control of a separate master process on the same machine. If the virtual site crashes, the master is notified, but is not otherwise affected.

5 Coordination

A perceived drawback of the coordination model is the use of a single shared space, which would introduce significant latency and reliability problems if applied to the Internet, as well as being very difficult to manage [98]. This has led to the introduction of multiple spaces to make the mapping to a decentralized architecture easier and scalable. Hierarchical spaces are also useful for representing nested domains with trust boundaries.

A major question is how these richer models should be integrated with existing Internet services. Various solutions include adding spaces as new services, masking Internet services as shared spaces, and introducing special purpose agents to mediate between the shared spaces and services [34].

Another trend has been the augmentation of the coordination laws of the shared space so that its behaviour can be modified. This has proved useful in an Internet setting for creating enforceable ‘social behaviour’ which malicious entities cannot bypass [51].

Several problems are only starting to be considered: the conflict between coordination (which encourages communication) and security (which restricts it), and partial failure.

Linda Linda supports a shared dataspace, called a tuple space, and a small set of operations for reading, adding, and removing tuples from the space [23].

Multiple flat tuple spaces were introduced quite quickly to facilitate distributed programming [63]. Hierarchical tuple spaces followed, most notably in a major revision to the Linda model by its authors, called Bauhaus Linda [24]. Bauhaus removes the distinction between tuples and tuple spaces, leading to a hierarchy based on multisets. Since tuple spaces are also tuples, they are first class citizens, and so can be moved around easily. The distinction between passive and active data is removed, making it simpler to reposition processes and copy them. Movement between spaces is based on short steps, either to the parent or one of the space’s children.

Linda extensions especially for the Web/Internet include JavaSpaces [113], PageSpace [31], and WCL [92].

SeCoS [19] supports secure spaces which allows tuples to be locked with a key. A matching key for unlocking can be passed to other processes. This is quite similar to public key cryptography.

Another popular security device is to assign access rights (e.g. read and write permissions) to tuples. Menezes et al. [92] suggests including group permissions so that capabilities can be supported.

5.1 Linda and LP

Ciancarini [30] suggests four approaches to creating LP coordination models.

The first is to add the Linda primitives and shared dataspace into Prolog. The typical mapping is to represent tuples by terms, and replace pattern matching with unification. Backtracking is not supported for the reading, writing, and removal operations. Terms added to the dataspace are copies, so it is not possible to create dynamic communication links by including variables in the terms shared with other processes.

Several Prolog systems include Linda-style libraries, including BinProlog [16] and SICStus Prolog [110]. The SICStus library has been used widely as an implementation tool (e.g. [97, 114]).

An operational semantics for a Linda-like coordination language using ground terms as tuples and unification for pattern matching is presented in [96].

The second approach is a variation of the first, where backtracking is allowed over the primitives. Functionality of this kind is very complex to implement, and difficult to reason about, which may explain why it has never been utilised in a coordination setting.

A third alternative is to create a new abstract machine for Prolog based on the Linda model. This line of research has not been investigated to date.

A fourth possibility is to permit the logic language to use the concurrent process model inherent in the Linda style of programming. Ciancarini and others have concentrated on this approach, resulting in Shared Prolog and its descendents.

In the following we survey the work in the first and fourth categories relevant to Internet programming.

μlog and its Relatives μlog supports a tuple-based dataspace/blackboard [73]. No distinction is made between passive and active data, the latter being executed as goals, allowing processes to be dynamically created.

The operational and denotational semantics of μlog are investigated in [46, 48]. Subsequent versions of the language added multiple named blackboards, primitives with optional guards/constraints, and bulk operations [48].

μ^2log introduced distributed blackboards, and virtual boards as local 'aliases' for boards on remote hosts [47]. Boards are treated as another form of data by the Linda primitives, which allows them to be moved between locations easily.

An operational semantics for μ^2log is outlined in [47].

Multi-BinProlog μ^2log was a strong influence on the design of Multi-BinProlog [49]. It implements the virtual board device using RPCs: a local process transparently communicates with a remote RPC server representing the remote board. It carries out the board operations locally, using a dedicated thread for the request.

LogiMOO LogiMOO is a virtual world framework which rests on top of Multi-BinProlog [120, 118]. It supports the notions of places, objects, and agents using the underlying boards and threads of Multi-BinProlog. A place corresponds to a board at a remote site, objects to terms and URLs of Web pages and multimedia, and agents are collections of threads.

BinProlog's binarization preprocessor has been extended to support first order continuations, which are the basis of mobile threads programming [119].

Jinni The on-going integration of BinProlog and Java led to Jinni (the Java INference engine and Networked Interactor) [117]. It uses Prolog engines (coded in Java) to execute goals, with each engine in a separate thread. If a thread wants to communicate with a remote board it must move to its place by utilising first order continuations and socket links.

A suggested Jinni solution to malicious hosts is to have a thread take its own interpreter with it when it moves. This is feasible due to the small size of Jinni's Prolog engine. Also, returning threads can be checked since they are first order entities.

Shared Prolog and its Descendents Shared Prolog supports the creation of parallel processes consisting of Prolog programs extended with a guard mechanism [18]. The processes communicate via a shared dataspace of terms, using unification for matching.

ESP and Polis Extended Shared Prolog (ESP) [20] is based on the Polis model [30, 32] which introduced the notion of multiple tuple spaces and the storage and manipulation of rules inside the dataspaces as first-class entities (called program tuples).

A tuple space (called a place or sometimes a theory) is represented by a named multiset of tuples; operations are annotated with the place name where they are to occur. Names can be freely passed around inside tuples, so allowing dynamic reconfiguration. Places can be dynamically created by a process.

The following is a Polis theory called `eval_f` which consumes a tuple `tuple(Input)`, calls `f/4`, then produces a result `tuple(Ouput)` and recurses.

```
theory eval_f(State) :-
eval
    { tuple(Input) }  -->                 % consume
    f(input, State, Output, NewState)     % process
    { tuple(Output), eval_f(NewState) }   % output
with
   f(I, S, O, NS) :- ...       % Prolog defn
```

Later versions of the language allowed spaces to be distributed [30]. This was implemented on top of a network version of Linda (Network-C-Linda) that supports clusters of workstations

The Polis model was further extended to support hierarchies of multiple tuple spaces [32]. It uses a naming mechanism similar to Bauhaus Linda to allow a multiset to refer to its parent and children.

An operational semantics for Polis and a formal semantics using a temporal logic of actions are presented in [32]. An alternative approach is to specify place behaviour using a chemical interpretation in which 'molecules' float, interact, and change according to reaction rules [29].

ACLT and Tuple Centers Processes in ACLT (Agents Communicating through Logic Theories) interact with named logic theories (tuple spaces holding Prolog clauses) [97]. One mode of interaction is via Linda operations, which may cause the theory to change. The other mode uses demo-style predicates to execute goals against the theory. This means that ACLT spaces can be interpreted either as communication channels or as knowledge repositories.

Later work on ACLT [51] introduced a separate first order logic language called ReSpecT (Reaction Specification Tuples). ReSpecT reaction rules are used by the ACLT coordination model to define the behaviour of its theories in response to communication operations. Logical theories are now called tuple centers because of their programmable coordination feature.

For instance, the following reaction

```
reaction(out(p(_)), (
   in_r(p(a)), in_r(p(X)), out_r(pp(a,X)) ))
```

is triggered whenever a new `p/1` tuple is inserted into the tuple space. Its intended effect is to replace two `p/1` tuples (one of them should be `p(a)`) with a single `pp/2` tuple.

This approach has the advantage (and disadvantage) of allowing the normal meaning of a Linda operation (e.g. an `out`) to be varied. The most important benefit is that additional coordination logic can be located in the tuple center itself, augmenting the standard behaviours of operations. This makes it easier to specify enforceable global coordination behaviours for processes.

Another advantage of reaction rules is that less communication operations are required to implement nonstandard protocols (which is important in a network setting). Rules can also support security features such as the cancellation of an illegal agent operation.

TuCSoN TuCSoN (Tuple Centers Spread over Networks) extends the tuple center approach to the Internet [41]. The Internet is viewed as a hierarchical collection of locality domains [42]. A mobile entity must dynamically acquire information about the location of resources and their availability (to that entity) as it moves over the network. There may be parts of the network with restricted access, which requires a means to authenticate entities and allocate them permissions.

Tuple centers are located on Internet nodes, and are used as the building blocks for coordination, and for resource control and access. An Internet domain is defined in terms of a gateway which controls access to places inside the domain and to subdomains gateways.

The following example outlines how an agent might explore a domain inside TuCSoN:

```
<goto d>                % migrate to gateway d
<identify>              % authenticate agent with d
?rd(places)             % get places info
?rd(commspaces)         % get tuple centres
<for pl in places do>
  <goto pl>             % visit place pl
  <for tc in commspaces do>
    tc?op(tuple)        % ask tuple centre tc of place pl
                        % to execute op(tuple)
```

5.2 Parallel Multiset Rewriting

Parallel multiset rewriting gained wide notice in Gamma [15]. A program is a collection of condition/action rules which employ a locality principle – if several conditions hold for disjoint subsets of the multiset being processed then the actions can be carried out in parallel.

One of the benefits of this programming style is its chemical solution metaphor. The multiset is the solution, and rules specify chemical reactions which work on distinct parts of the solution, changing sets of molecules into others [14].

LO Linear Objects (LO) [10] was originally proposed as a merger of LP and object oriented programming. It extends Prolog with multi-headed formulae which work on a multiset of terms. OR-parallelism can be used for rule evaluation, which implies the atomic removal of terms from the multiset. Later versions of LO include the ability to clone new multisets and to terminate a multiset [9]. A broadcast operator can send a copy of a term to all the multisets.

Around this time, Interaction Abstract Machines (IAMs) were developed as a model for concurrent multi-agent worlds [9]. The IAM can be used as a computation model for LO.

CLF CLF (Coordination Language Facility) utilises parallel rewrite rules as a scripting language for specifying and enacting the coordination of distributed objects [8].

A typical coordination action consists of the atomic removal of a number of resources from some objects, followed by the insertion of resources into other objects. This atomic removal and insertion is captured succinctly by rewrite rules.

The objects refereed to in the rules can contain sophisticated protocols for extracting resources from distributed services (e.g. based on negotiation, atomic performance). The objects also locate suitable services, which may be distributed over the Internet.

6 Message Passing

Most LAN and Internet-based message passing languages utilise existing communication protocols (e.g. TCP/IP) or libraries (e.g. PVM). This supplies useful functionality immediately, such as guarantees of service in TCP and failure detection in PVM, as well as support for interaction across heterogeneous platforms and processes written in varied languages. However, there are several disadvantages, related to the mismatch between the LP paradigm and the low-level (imperative) viewpoint offered by these protocols and libraries.

Invariably, the LP level must restrict the use of logic variables in messages in order to maintain the uni-directional nature of the message passing at the lower levels. This usually means renaming variables as they are transmitted, so eliminating the possibility of dynamic communication channels. Backtracking is also ruled out since the underlying primitives are deterministic. Some libraries, such as early versions of MPI [62], do not support dynamic process creation, which naturally affects process support in the LP layer

The typical unit of computation is a sequential Prolog process, which promotes a coarser grained parallelism than in concurrent LP. This may be a benefit since it makes it easier to justify the communication costs of distributing tasks between machines. Several languages also support threads, often for handling subtasks inside a process.

6.1 IC-Prolog II and Friends

IC-Prolog II can create concurrently executing Prolog threads, possibly spread over separate machines [28]. Threads are independent, sharing no data, but can communicate using pipes if the threads are on the same host, or with primitives utilising TCP/IP if the threads are on different machines. The messaging operations are non-backtrackable and send/receive terms with variables renamed.

A concurrent server example using TCP:

```
conc_server(Port) :-
    tcp_server(Port, Socket),    % listen for connections
    multi_serve(Socket).

multi_serve(Socket) :-
    tcp_accept(Socket, New),     % get a connection
    fork( service(New) ),        % create thread to service it
    multi_serve(Socket).         % look for more connections
```

Processes can employ mailboxes; a message is sent to a mailbox by referring to its ID or name, which is globally unique. Mailboxes can be linked so that messages placed in one box are automatically copied to other boxes. Links allow the creation of arbitrary communication topologies.

April April is a concurrent language, offering distributed objects as lightweight processes [88]. Each process has a globally unique handle, which can be registered with DNS-like April name servers. The communications model is point-to-point, being based on TCP/IP. A receiving process can use pattern matching to search for a suitable incoming message stored in a message buffer.

A server which executes a task:

```
server([any]{}?T) {
  repeat {
    [do,any?arg] -> {      % request a task using arg
       T(arg);             % execute it
       done >> replyto     % reply when finished
    }
  } until quit             % server continues until a quit
};
```

The server is forked and its name (`server1`) is registered with the local name server:

```
server1 public server(taskName)
```

Processes can send messages to the server at its location (foo.com for example):

```
[do,taskArg] >> handle?server1@foo.com
```

April combines several paradigms, including LP. It includes a variety of data types based on tuples and sets, higher-order features such as lambda, procedure and pattern abstractions, real-time support, and an expressive macro language.

Qu-Prolog Qu-Prolog has been extended with April-like communication support and multi-threading [38]. Threads located on the same machine can utilise Linda-style operations on a shared dynamic database.

Qu-Prolog extends Prolog with support for qualified terms, object variables, and substitutions, which allows it to easily express inference rules for many kinds of logics, and implement theorem provers efficiently. These features are particularly suited for building agent systems.

QuP++ is an object oriented layer on top of the Qu-Prolog/April work [37]. It offers a class-based language with multiple inheritance, where a class is a collection of static (unchangeable) and dynamic clauses and state variables, and an object is a collection of one or more independently executing threads. Synchronous and asynchronous remote method calls are available, including a form of distributed backtracking.

Go! Go! is a higher order (in the functional programming sense), multi-threaded LP language, making use of April's symbolic message communication technology [36]. Go! does not support Prolog's meta features for interpreting data as code; instead of `assert` and `retract`, Go! has a restricted assignment mechanism.

6.2 PVM-Prolog

PVM-Prolog can create distributed Prolog processes communicating by message passing [43]. It differs from the other Internet-based languages described here in that it uses the PVM messaging library [61].

The principle advantages of PVM over TCP/IP are that it offers a higher level communications layer (messages rather than byte streams), a virtual machine, and plentiful commands for process/resource management (e.g. for fault tolerance).

PVM-Prolog has two components – the Parallel Prolog Virtual Machine (PPVM) and a process engine (PE). The PPVM acts as a LP interface to PVM; for instance, it supports Prolog terms as messages. A PE executes Prolog processes, and several may be created on a single virtual machine.

In a later version of PVM-Prolog, threads were introduced to support fine-grain concurrency within a process [107]. Each thread represents an independent query over the process' database, but they can interact via shared term queues.

6.3 Mercury

Mercury is a pure logic/functional programming language, utilising a strong type system with parametric polymorphism and higher-order types, mode declarations on predicates, extra determinism information, and a module system.

MCORBA is a binding to the CORBA distributed object framework for Mercury [74]. The approach is made possible by utilising Mercury's type classes and existential types. A type class offers a form of constrained polymorphism which is quite similar to a Java interface – the class is specified in terms of method signatures which can be instantiated later. An existential type allows a predicate to return a variable whose type is constrained to be of a particular type class but not an actual concrete type instance. This is useful for supporting CORBA functions that return generic objects which are later 'narrowed' to a specific object type.

A compiler back-end is being developed for Mercury which targets Microsoft's .NET Web services framework. This will allow components coded in Mercury to inter-operate with other .NET elements programmed in C#, Visual Basic, C++, and other languages. Preliminary details are available at
`http://www.cs.mu.oz.au/research/mercury/information/dotnet/mercury_and_dotnet.html`.

7 Client-side Execution

Sections 7 and 8 are complementary since they describe client/server mechanisms, which are still the most common way of using the Web (e.g. a Web browser communicating with a server).

In general, client-side code can be more closely integrated with the browser and so can offer more sophisticated user interfaces than server-side solutions. Also, once client-side code is downloaded it does not need to communicate with the server, thereby avoiding networking problems that can affect server-side applications. Only the code which is needed for the current task has to be downloaded and, since it is a copy of the original, it can be changed or combined with other code without affecting the original.

A disadvantage of the client-side approach is security when running foreign code locally.

We consider four topics: libraries, query languages, Web pages as programs, and the combination of Java and LP. Parts of this section previously appeared in [83].

7.1 Libraries

Most modern Prologs contain libraries for creating TCP and UDP sockets (e.g. SICStus, BinProlog, Amzi, LPA, Quintus). With these it is possible to code support for protocols such as HTTP and NNTP. System calls to `telnet` can also be employed as a foundation for network functionality.

PiLLoW PiLLoW is the most elaborate Web library [21]. The main client-side predicate, fetch_urls/2, downloads a page corresponding to a URL with additional options specifying such things as a timeout limit, the maximum number of retries before the predicate fails, and user ID and password details for protected sites. PiLLoW's parsing predicates extract HTML tag attributes and values from a page string, returning them as Prolog terms. It is possible to convert an entire page into a list of terms, making it more amenable to manipulation.

The following call fetches two documents, also getting the type and the size of the first, and checking for non-fatal errors in the second, and allowing only one socket to be used:

```
fetch_urls([ doc('http://www.foo.com',
  [content(D1), content_length(S1), content_type(T1)] ),
             doc('http://www.bar.com/drinks.htm',
  [content(D2), errors(non_fatal,E)] )    ],
           [sockets(1)]
          ).
```

Streaming Download Davison [45] models page retrieval using a stream-based approach, in the context of a concurrent LP language. download/4 returns a page incrementally as a partially instantiated list (stream) of characters, and includes Parlog meta-call arguments for status reporting and control. download/4 can be used as a building block for AND- and OR- parallel Web search, time-outs, repeated attempts to download, and the cancellation of slow retrievals.

A retrieval predicate with a time-out facility (using deep guards):

```
mode timeout(?, ?, ^, ^, ?).
timeout(_Time, Request, Text, Result, Stop) :-
  download(Request, Text, Result, Stop) : true.
timeout(Time, _, _, err(timeout), _) :-
  sleep(Time) : true.
```

`download/4` executes the request and returns the text of the page, unless it is stopped or the second clause succeeds because the timeout has expired.

Webstream, a macro extension to April, also views Web page downloading as incremental stream-based retrieval [69]. It extends the idea with a pipelining mechanism (reminiscent of UNIX pipes) which allows stream data to be filtered concurrently.

7.2 Query Languages

A more abstract approach to client-server computation is to view the Web as a vast heterogeneous collection of databases, which must be queried in order to extract information.

In fact, in many ways the Web is *not* similar to a database system: it has no uniform structure, no integrity constraints, no support for transaction processing, no management capabilities, no standard query language, or data model.

Perhaps the most popular data model for the Web is the labelled graph, where nodes represent Web pages (or internal components of pages) and arcs correspond to links. Labels on the arcs can be viewed as attribute names for the nodes. The lack of structure in Web pages has motivated the use of semistructured data techniques, which also facilitate the exchange of information between heterogeneous sources.

Abiteboul [1] suggests the following features for a semistructured data query language: standard relational database operations (utilising a SQL viewpoint), navigational capabilities in the hypertext/Web style, information retrieval influenced search using patterns, temporal operations, and the ability to mix data and schema (type) elements together in a query.

Many languages support regular path expressions over the graph for stating navigational queries along arcs. The inclusion of wild cards allows arbitrarily deep data and cyclic structures to be searched, although restrictions must be applied to prevent looping.

Query Computability The question of query computability is considered in [3, 91]. Mendelzon and Milo [91] focus on two aspects that distinguish Web data access from database manipulation: the navigational nature of the access, and the lack of concurrency control. They investigate the Web Calculus, a query language quite similar to WebSQL, under the assumption that the Web is static (no updates), and the more realistic dynamic case.

The main open problem is how to characterize queries which definitely terminate. One sufficient condition is to avoid the use of the regular expression * pattern in paths.

Abiteboul and Vianu [3] differ from Mendelzon and Milo in assuming that the Web is infinite, an assumption which seems somewhat dubious. They examine the computability of first order logic, Datalog, and Datalog with negation.

Queries Languages for HTML Pages

Relational WebSQL models the Web as an extremely large relational database composed of two relations: Document and Anchor [12]. Document contains one tuple for each Web document, and the Anchor relation has one tuple for each anchor in each document.

An interesting feature of the language is its use of regular expressions involving URL links to define paths between documents. For example, '->' denotes a link between two pages at the same site, while '=>' is a link to a page at another site. Chains of links (called *path regular expressions*) are created using sequential, alternating, and multiplicative operators.

For example, suppose we want to find a tuple of the form `(d,e)`, where `d` is a document stored on our local site (`http://www.foo.com`), and `e` is a document stored elsewhere. The query to express this is:

```
SELECT d.url, e.url
FROM   Document d SUCH THAT  "www.foo.com" ->* d,
       Document e SUCH THAT d => e
```

`d` is bound in turn to each local document, and `e` is bound to each document directly reachable from `d`.

Datalog, F-logic WebLog utilises a Datalog-like language to retrieve information from HTML documents and to build new documents to hold the results [80]. It represents links as first class entities, specified using molecular formulas (somewhat like F-logic terms). A molecular formula for a URL lists attribute/value pairs for the relevant data inside the page. Attributes can be keywords, page substrings, links or tags. Backtracking allows alternative matching URLs to be found.

This example collects all the links in the page `http://www.foo.com`, and retrieves the titles of the pages pointed to by the links. It stores the results in a new Web page `ans.html`.

```
ans.html[title->"All Links", hlink->>L, occurs->>T] <--
  http://www.foo.com[hlink->>L],            % a molecule
  href(L,U), U[title->T].
```

FLORID is a prototype deductive object oriented database using F-logic, containing builtins for converting Web documents and their links into objects [86].

The FLORID database corresponds to a labelled graph where nodes are logical object IDs (OIDs) and object and class methods are labelled edges.

FLORID provides general path expressions, very similar to those in Lorel (discussed below), which simplify object navigation and avoids the need for explicit join conditions. The operations (e.g. *, +, ?, |) are specified and implemented in F-logic, making them amenable to simplification rules and to being extended (e.g. with path variables).

Web documents are modeled by two classes: `url` and `webdoc`. `url` represents a link and has a `get()` method for retrieving the referenced page as a `webdoc` object. The `webdoc` class has methods for accessing the text in a page, its links, and various meta-level details.

The program below collects the set of Web documents reachable directly or indirectly from `http://www.foo.com` by links whose labels contain the string "database".

```
("www.foo.com":url).get.(Y:url).get <-
  (X:url).get[ hrefs@(L) =>> {Y} ],
  substr("database",L).
```

The Web->KB Project The Web->KB Project [40] demonstrates the potential of using machine learning techniques for extracting knowledge bases from Web sites. The best approach uses a combination of statistical and relational rule learners; for example, a Naive Bayes text classifier combined with FOIL. This merger is well suited to the Web/hypertext domains because the statistical component can characterise text in terms of word frequency while the relational component can describe how neighbouring documents are related by hyperlinks [111].

Labelled Graph Models Much of the present research on semistructured data query languages centers on object models for edge-labelled directed graphs. Little of the work is logic-based, but the proposals embodied by these languages are sufficiently important to strongly influence logic programming approaches. To some extent this can be seen in FLORID.

Lorel is a query language in the style of SQL and OQL [2], originally used within the TSIMMIS system (discussed in section 9.1). Its object model, OEM (Object Exchange Model), is an extension of the structured object database model ODMG.

Lorel offers regular path expression, with wild cards and path variables, for specifying navigation over the graph. Path expressions extend Lorel's functionality beyond that of SQL and OQL.

The following Lorel query finds the names and zipcodes of all the "cheap" restaurants in a `GoodFood` database.

```
SELECT GoodFood.restaurant.name,
       GoodFood.restaurant(.address)?.zipcode
WHERE  GoodFood.restaurant.%grep "cheap"
```

The "?" makes the address part optional in the path expression. The wildcard "%" will match any sub-object of restaurant, which is then searched with `grep` to find the string "cheap".

Queries to pages using XML or RDF XML (eXtensible Markup Language) is a notation for describing labelled ordered trees with references [130]. It is possible to type portions of XML data with DTDs (Document Type Definitions).

A DTD defines the types for elements, and what attributes can appear in each element. The specification is written using regular expressions.

Specifying a query language for XML is an active area of research, much of it coordinated through a W3C (the World Wide Web Consortium) working group [127]. The suggested features for such a language are almost identical to those for querying semistructured data [56].

It is hardly surprising that most proposals utilise models which view XML as an edge-labelled directed graph, and use semistructured data query languages (e.g. Lorel). The main difference is that the elements in an XML document are sometimes ordered.

Relational Models Shanmugasundaram et al. [108] investigate the possibility of using a traditional relational database engine to process XML documents conforming to DTDs. Problems with query translation center on handling regular path expressions which frequently translate into many SQL queries and expensive joins. This suggests that while regular path expressions are high-level, they are also costly to execute in many cases.

Functional Programming Approaches XDuce is a tree transformation language, similar to functional programming languages, but specialized for XML processing [70]. It adds regular expression types and regular expression pattern matching, similar to pattern matching in ML. The result is that XML document fragments can be manipulated as XDuce values.

XMλ is a small functional language (very close to Haskell) which maps DTDs into existing data types [90]. Document conformance to a DTD becomes a matter of type correctness.

Declarative Description Theory The Declarative Description Theory (DDT) is an extended definite clause logic language where substitution is generalised to specialization [4]. Specializations permit language elements to have specific operations for variable expansion and instantiation.

DDT is utilised in [11] to define a specialization system for XML elements and attributes. This allows clauses to use variables which may be partially instantiated XML elements or attributes, containing further variables. Pattern matching and binding use the specialization operators to manipulate these variables according to their XML definitions.

RDF RDF (Resource Description Framework) is an application of XML aimed at facilitating the interoperability of meta-data across heterogeneous hosts [131].

The SiLRi (Simple Logic-based RDF interpreter) utilises an RDF parser (called SiRPAC) to translate RDF statements into F-logic expressions [50].

Metalog is a LP language where facts and rules are translated and stored as RDF statements [87]. Facts are treated as RDF 3-tuples, while rule syntax is supported with additional RDF schema statements for LP elements such as head, body, if and variable.

7.3 Web Pages as Programs

LogicWeb The LogicWeb system [83] resides between the browser and the Web. As pages are downloaded, they are converted into logic programs and stored locally.

A Web page is converted into several predicates, including facts holding meta-level information about the page, a fact containing the page text, and page links information. Programs can be composed together using operators inspired by work on compositional LP, implication goals, and contextual LP. There is a context switching operator which applies a goal to a program specified by its ID. If the program required by the goal in the context switch is not in the local store, then it is transparently downloaded. This behaviour hides low-level issues concerning page retrieval and parsing.

The query:

```
?- subject_page("LP", "http://www.foo.com", P).
```

will bind P to a URL which is related to "LP" and is linked to the starting page. `subject_page/3` is defined as:

```
subject_page(Subject, StartURL, URL) :-
  lw(get,StartURL)#>link(_, URL),
  lw(get,URL)#>h_text(Source),
  contains(Source, Subject).
```

The `lw/2` call retrieves the starting Web page using an HTTP GET message and selects a link URL. The source text of that page is examined to see if it contains "LP"; if it does not then a different link is selected through backtracking.

An extension of the LogicWeb system deals with security aspects of client-side evaluation [84] – LogicWeb code can be downloaded with a digital signature. This is decrypted using the page's public key in order to authenticate the code. The decrypted signature is also used as an ID to assign a policy program to the code during its evaluation.

A concurrent LP version of LogicWeb is outlined in [45]: it allows queries to be evaluated using parallel versions of the composition operators. It utilises the download/4 operator described in section 7.1, and requires an atomic test-and-set primitive so that the client-side program store is updated atomically.

W-ACE and WEB-KLIC The concurrent constraint-based LP language W-ACE has explicit support for Web computation [101]. Some of its novel ideas include representing Web pages as LP trees and the use of constraints to manipulate tree components and the relationship between trees.

The following predicates extend the current HTML graph (`GraphIn`) with all the documents linked to `Page`, resulting in `GraphOut`.

```
update(Page, GraphIn, GraphOut) :-
  Links = { X : ref(X) <= Page},   % make a links set
```

```
  addLks(Links, Page, GraphIn, GraphOut).

addLks(0, _, GIn, GOut) :- GOut = GIn.
addLks(X:Rest, Page, GIn, GOut) :-
  get_url(X,Tree), !,      % get the page tree for X
  GNew = {(Page,Tree)} U GIn,
  addLks(Rest, Page, GNew, GOut).
addLks(X:Rest, Page, GIn, GOut) :-
  GNew = {(Page,dead)} U GIn,     % dead link info.
  addLks(Rest, Page, GNew, GOut).
```

W-ACE also contains modal operators for reasoning about groups of pages, and composition operators very similar to those in LogicWeb.

The authors of W-ACE have been working on a Web version of the concurrent LP language KLIC, called WEB-KLIC [98]. Their primary goal has been the augmentation of its CGI facilities (i.e. for server-side computation).

7.4 Java and LP

A spate of Java/LP systems have appeared in recent years. A connection to Java gives an LP language immediate access to a very wide range of classes for GUIs, imaging, multimedia, business components, and networking support. However, there are some serious disadvantages, the main one being the mismatch between the Java programming model (imperative/object oriented) and LP, which occurs in all multi-paradigm approaches. For instance, how should traditional control flow be combined with non-determinism, how should destructive assignment be reconciled with logic variables, and how are the variety of data structures/types/classes in Java mapped to atoms and terms? How should garbage collection be handled in a hybrid environment?

Calejo [22] categories Java and LP systems into two broad camps: "Prolog in Java" and "Prolog+Java". URLs for most of the current systems can be found at his Web site: `http://dev.servisoft.pt/interprolog/systems.htm`.

Prolog in Java The 'in' crowd can be divided into those systems that compile Prolog code into Java (e.g. jProlog, LLPj, MINERVA), and those that utilise a Prolog interpreter as a Java class (e.g. BirdLand Prolog, DGKS Prolog, JavaLog, Jinni, W-Prolog).

A benefit of "Prolog in Java" is the close integration, which permits Prolog code to more directly employ Java functionality, and be downloaded to browsers alongside Java applets.

Prolog+Java The "Prolog+Java" camp consists of systems which link Prolog and Java via their foreign language interfaces (e.g. Amzi! Prolog, Jasper, JIPL, JPL, NanoProlog), and systems which use a network link (usually socket based), such as InterProlog.

The following fragment shows how the Jasper package in the SICStus library can be used by Java. A Prolog query `connected("Wilmslow","Stockport", Route)` is constructed, passed to the Prolog engine which applies it to the program in `train.ql`, and all the different possible routes are printed back on the Java side.

```
SICStus sp = new SICStus(argv,null);   % Prolog engine
sp.load("train.ql");

SPPredicate pred = new SPPredicate(sp, "connected", 3, "");
SPTerm to = new SPTerm(sp, "Wilmslow");
SPTerm from = new SPTerm(sp, "Stockport");
SPTerm route = new SPTerm(sp).putVariable();

SPQuery q = sp.openQuery(pred,
             new SPTerm[]{from, to, route});   % build query
while (q.nextSolution())
  System.out.println( route.toString() );
```

An obvious disadvantage of "Prolog+Java" over "Prolog in Java" is the requirement to have two distinct systems running for any application. This makes programs harder to write, debug, maintain, and complicates portability.

CCJava A somewhat different perspective on combining Java and LP is embodied in CCJava [112]. As mentioned earlier in the section on Distributed Oz (section 4.2), it adds single assignment variables and statement-level threads to Java as a way of enhancing its thread communication features. This permits a Java program to use techniques such as incremental and back communication popularized in concurrent LP.

8 Server-side Execution

Server-side evaluation typically involves the user in completing a form on their browser, which is submitted across the network to a Web server to be processed. The most widespread server-side evaluation mechanism is the Common Gateway Interface (CGI) which delivers form details to programs, and routes any output from the code back to the user.

In general, server-side software is ideal for controlling resources such as databases which cannot be sent over the Web for various reasons. Also, having all users communicate with a central location makes it easier to program applications with more complex communication requirements, such as chat systems or market places.

One disadvantage of server-side programming is the difficulty of extending the user interface. For instance, it is not possible to intercept the activation of a hypertext link or to augment the forms interface with additional GUI elements.

Also, since server-side scripts are usually located on different machines from the forms which use them, communication latency can be a problem. A further drawback is the load on the server caused by multiple clients running scripts.

Parts of this section previously appeared in [83].

8.1 Libraries

There are many libraries which enable Prolog programs to process information from CGI input, and generate suitable replies (typically, new Web pages) [7, 21, 85].

The following server-side program uses the PiLLoW library [21] to extract a name from CGI input, call a user-defined `response/2` predicate to get an answer, and then construct a Web page.

```
main(_) :-
  get_form_input(Input),        % read CGI input
  get_form_value(Input, person_name, Name),
  response(Name, Response),     % lookup response
  output_html([ form_reply, start,
      title('Response'),
      heading(2, 'Response'),
      Response, end ]).
```

`get_form_input/1`, `get_form_value/3`, and `output_html/1` are PiLLoW library predicates.

8.2 Modifications to CGI

A CGI script is newly invoked for each query from a client, which can be a problem if the script has to load very large support software. Much of this overhead can be avoided by using shared dynamically linked libraries, and utilising compilers which generate fast object code and small executables. Also, it is far from clear whether the poor performance of a particular Prolog CGI script is due to its coding in Prolog, or because of network and machine overheads, and/or the slowness of CGI.

A related issue is that the client-server model allows a server to process several clients concurrently, which implies that several invocations of the same script may need to be running simultaneously. This may not be practical because of the size of the system, and also makes changes to shared resources more complicated.

Separating Interface and Process Another server-side solution is to separate query processing into two parts: a light-weight CGI script which acts as an interface to a separate heavy-weight task process. A key feature of the task process is that it is continually running, and so only needs to be loaded once. In the context of LP, this process might be a Prolog system or logic database.

This approach is used in the EMRM knowledge base of medical records, which utilises the OR-parallel Aurora system to process multiple queries at once [115].

The PiLLoW/CIAO library supports a higher level communications layer between the interface and task processes based on Active modules. Each invocation of the interface script communicates with the task process as if it was calling a module [21]. The authors speculate on using &-Prolog/CIAO to parallelise their Prolog engine.

Another problem, addressed in the EMRM system, is how to deal with lengthy browser interactions, which require the task process to suspend while the user enters further details. A related difficulty, peculiar to LP systems, is how to deal with backtracking to a previous stage in the user interaction. The ProWeb system [85] records the pages associated with earlier stages, and can redisplay them as required. Backtracking may also make it necessary to rollback changes to (shared) resources. These problems can occur with any multi-user LP application, but are compounded by the forms-based user interface supplied by CGI, and the stateless nature of the HTTP protocol.

Replacing the Server A third server-side technique is to completely replace the traditional Web server by software which combines the functionality of a server with the particular task.

A notable LP solution in this style is the ECLiPSe HTTP server library, which allows a basic server framework to be customized for different communication protocols [17]. Indeed, the major advantage of this technique is the way that the server can be specialized for specific applications and communication modes. The main drawback is the large amount of work required to implement a fully featured server with concurrency control, error handling, administrative tools, and so on.

9 Integration of Heterogeneous Data Sources

The issues related to data integration on the Web/Internet are similar to those for integrating heterogeneous database systems, but are arguably more complex due to the large number, and evolving nature, of Web sources, the lack of meta-data (i.e. schema) about the sources, and the degree of source autonomy. Semantic heterogeneity – the representation of the same or overlapping data in two or more ways – is a difficult problem.

Hull [71] identifies a number of data management architectures: mediation systems (integrated read-only views of data), mediation with updates (which introduce the view update problem, federated systems, and work flow architectures.

Research on Web data integration focuses on mediation, and borrows from the DARPA I3 reference architecture [72]. Wrappers are employed at the lowest level to translate between a Web source's local language, model and concepts and the global concepts embodied by the system. Mediators obtain information from the components below them (which may be wrappers or other mediators).

The wrapper and mediator architecture has two distinguishing features over integrated heterogeneous databases: a mediator does not directly communicate with a source, instead interacting with its wrapper, and a user does not pose queries in the schema of the data sources, instead using the mediated, global schema. This last point requires the mediator to reformulate queries using some kind of source description.

There are two main approaches to specifying a mediated schema and its reformulation, which Hull terms "classical integration" and the "new wave" [71].

Classical integration defines global, mediated schema as views over the local schema. Query reformulation becomes very simple – view unfolding or partial evaluation until the query is expressed in terms of the local schema elements. A survey of this approach, as applied to heterogenous databases, can be found in [93]. Many Web-based systems use classical integration (e.g. TSIMMIS [100], described below).

The new wave considers global schema to be independent of the local schema to a large extent. Data held at the sources are expressed as views over the chosen global schema, to specify how mediated schema relations are to be translated. An advantage is the ease of adding/removing sources since they do not require a view mechanism to be altered at the mediator level.

Many new wave systems utilise description logics as glue between local and global schema (e.g. Information Manifold [81], described below). A description logic is typically a subset of first order logic with specialized syntax that makes it suitable for describing and reasoning about entities and relationships. It is often combined with a Datalog-based query language which handles other aspects of inference.

Two good sources of papers on this topic are the 1999 Workshop on LP and Distributed Knowledge Management [99], and the recent JLP special issue on logic-based heterogeneous information sources [104].

9.1 Classical Integration

TSIMMIS TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources) implements a mediator hierarchy, where a mediator may converse with sub-mediators or with wrappers [100]. The system concentrates on the querying of semistructured or unstructured data, and utilises the OEM (Object Exchange Model) data model. The query language is MSL (Mediator Specification Language), which is also used to describe mediators and wrappers in various ways.

MSL is an object logic with a Datalog-like syntax. Mediators are specified using MSL rules. Each rule maps a set of objects at a source into a 'virtual' object at the mediator. Mediator objects with the same OID are fused together in ways specified by the rules. Mediator rules are like a database view since the sources are only queried for objects when a query arrives.

For example, the rule *paper* is defined as:

```
<paper { <title T><author A><abstract B><conf C> }> :-
  <entry { <title T><author A><abstract B> }>@s1,
  <entry { <title T><conf C> }>@s2.
```

paper is essentially a join of the views exported by sources s1 and s2, with `title` being the join attribute. The head consists of an OEM object with the label `paper`, and a list of sub-objects describing the `title`, `author`, and so on.

MSL without negation and OIDs can be considered a variant of Datalog. Full MSL can be converted to Datalog with function symbols and negation.

A query is evaluated by expanding the applicable mediator rules using unfolding until the query is expressed in terms of source information only. This can lead to exponential growth in the rule set as m query conditions unify with n rules to produce n^m expanded rules. TSIMMIS employs a range of techniques to limit such growth.

Medlan The Medlan system [13] implements a declarative analysis layer on top of a commercial GIS. The layer consists of multiple logic theories which can be composed together using meta-level operations, which form a program expression for the resulting collection of theories. A goal can be executed against a program expression. The operations in Medlan and LogicWeb (discussed in section 7.3) are similar.

9.2 New Wave Integration

Information Manifold New wave integration is typified by Information Manifold [81] which provides uniform access to a heterogeneous collection of more than 100 information sources, most of them on the Web. Its query language is based on a dialect of the description logic CARIN, which offers a fragment of first order logic almost equivalent to non-recursive Datalog. The Information Manifold architecture is based on global predicates, where each information source has one or more views defined in terms of those predicates.

Context Interchange The context interchange strategy primarily addresses the issue of semantic heterogeneity, where information sources have different interpretations arising from their respective contexts [64].

The global domain utilises the COIN data model; the COINL language offers a mixture of deductive and object oriented features, similar to those in F-logic.

Elevation axioms say how source values are mapped to semantic objects in the global domain. Context axioms include conversion functions which state how an object may be transformed to comply with the assumptions of a context. Conversion functions are crucial for allowing semantic objects to be moved between contexts.

An interesting feature of query rewriting in this approach is the use of abduction. Initially domain model axioms, elevation axioms and context axioms are rewritten as Horn clauses, with the addition of generic axioms defining the abductive framework and other integrity constraints. The abductive rewriting of the query is achieved through backward chaining until only source relations (and builtins) are left. Backtracking is used to generate alternative rewrites.

The abductive process is implemented using ECLiPSe and its Constraint Handling Rules (CHR) library. However, the authors remark on the suitability of Procalog for this purpose. Procalog is an instantiation of the work of Wetzel, Kowalski, and Toni on unifying abductive LP, constraint LP, and semantic query optimisation [129].

10 Conclusions

LP is a natural choice when a programming task requires symbolic manipulation, extended pattern matching, rule-based representation of algorithms, inference/deduction, a high degree of abstraction, and notation which reflects the mathematical basis of computation. It is not surprising that LP languages have found wide usage in the Internet domain.

A crucial requirement for Internet programming is a clear, unambiguous model of the Web/Internet. The most popular is the labelled graph, where nodes represent Web pages (or parts of them) and arcs correspond to links. This model can be directly translated into a LP framework, as seen for example in LogicWeb [83] and FLORID [86].

Logic variables are a powerful *and* efficient communications mechanism (e.g. see Janus [106], Strand [58], Distributed Oz [66]), and one which offers benefits to non-LP paradigms (e.g. see PCN [57], CC++ [27], CCJava [112]). One benefit is the possibility of using concurrent (constraint) LP programming techniques such as incomplete messages, bounded buffers, and short circuits.

(LP) coordination languages are sometimes discounted for being unable to encompass the size and complexity of the Web/Internet. Development of richer dataspaces based on multiple, hierarchical domains, the introduction of first order representations of tuple spaces, and the cross-fertilization of ideas from parallel rewrite systems has shown this view to be wrong. Coordination languages seem certain to play an important role in representing and controlling mobility, and in the integration of heterogeneous data sources.

Message passing languages illustrate the advantages of building on existing protocols and systems (e.g. TCP/IP, PVM, CORBA), and thereby gaining features like fault tolerance and reliable communication. The downside of using existing protocols is that it is difficult to retain LP elements such as logic variables.

Joining Java and LP is a growth area at present, although frequently driven only by the wish to gain access to Java's extensive libraries. However, CCJava shows how Java can benefit from logic variables [112], and Jinni is a close integration of Java and Prolog in a blackboard setting [117].

The server-side execution of LP code has been standardised into libraries which make it easy to develop CGI-based applications [21]. However, several issues remain concerning the suitability of combining LP and HTTP. One of these is the interaction between backtracking in the LP code and the 'backtracking' possible in a typical multiple forms interface.

Many query languages for semistructured data seem quite distant from LP, concentrating on relational database manipulation, information retrieval search techniques, temporal operators, and navigation based on regular path expressions. Nevertheless, Datalog and F-logic (and their many variants) are popular tools, although sometimes hidden behind SQL-like syntax (e.g. Lorel [2]).

There seems to be little LP involvement in the current development of query languages for XML. This contrasts with the rather active participation of the FP community, which have found interesting ways of using types to capture the regular expression aspects of DTDs.

Datalog and F-logic (and variants) are widely used for the integration of heterogeneous data sources, and there is much work to be done on applying results from heterogeneous database integration to the semistructured domain. Of particular note is the use of abduction in the context interchange strategy [64], and the deployment of inductive LP ideas for information discovery and data mining [40]. Also, as mentioned above, there seems scope for the application of LP coordination languages.

Failure handling is a difficult problem in an Internet setting; Sandra [54] and Distributed Oz [66] both offer a range of practical proposals. Approaches based on the meta-call also are worth further investigation.

Security is another under-developed topic, although we noted the use of read-only variables from FCP [94], and the meta-interpreter for client-side security in LogicWeb [84]. An approach with great flexibility is the tuple center [41] for enforcing 'social behaviour' as coordination laws.

The importance of mobility is being increasingly recognised: Distributed Oz takes an implicit view [66], while languages like TuCSoN [41], Jinni [117], and TeleLog [116] make mobility explicit. It appears that a visible notion of mobility is suited to most Web/Internet applications, where resources have stated locations.

One thread running through this paper is the utility of meta-level ideas. For example, meta-level mechanisms are used to specify security in LogicWeb, which can be viewed as a more expressive way of defining Java-like 'sandbox' restrictions [84]. Meta-level programming is at the heart of layered program approaches (e.g. in Logix [68]) and in process-to-processor mapping notations. Meta-level features often facilitate language extensions.

One Internet trend if the rise of components (e.g. ActiveX controls, JavaBeans) as the building blocks of applications. A key element of the component architecture is *reflection* – the ability of a component to manage its own resources, scheduling, security, interaction, and so on. Meta-level concepts are central to reflection, and so LP seems an ideal way of building such components.

References

1. Abiteboul, S. 1997. "Querying Semistructured Data", In *Proc. of the Int. Conf. on Database Theory (ICDT)*, Delphi, Greece, January.

2. Abiteboul, S., Quass, D., McHugh, J., Widom, J., and Wiener, J. 1997. "The Lorel Query Language for Semistructured Data", *Int. Journal on Digital Libraries*, Vol. 1, No. 1, April, pp.68-88.
3. Abiteboul, S. and Vianu, V. 1997. "Queries and Computation on the Web", In *Proc. of the Int. Conf. on Database Theory (ICDT)*, Delphi, Greece, January.
4. Akama, K. 1993. "Declarative Semantics of Logic Programs on Parameterized Representation Systems", *Advances in Software Science and Technology*, Vol. 5, pp.45-63.
5. Alkalaj, L., Lang, T., and Ercegovac, M. 1990. "Architectural Support for the Management of Tightly-Coupled, Fine-Grain Goals in Flat Concurrent Prolog", In *Int. Symp. on Computer Architecture*, June, pp.292-301.
6. Alouini, I. and Van Roy, P. 2000. "Fault Tolerant Mobile Agents in Mozart", Available at `http://www.mozart-oz.org/papers/abstracts/asama2000.html`.
7. Amzi! Prolog. 1996. "Internet and Web Tools", Available at `http://www.amzi.com/internet.htm`.
8. Andreoli, J.-M., Arregui, D., Pacull, F., Riviere, M., Vion-Dury, J-Y., Willamowski, J. 1999. "CLF/Mekano: A Framework for Building Virtual-Enterprise Applications", In *Proc. of EDOC'99*, Mannheim, Germany, September.
9. Andreoli, J.-M., Ciancarini, P., and Pareschi, R. 1993. "Interaction Abstract Machines", In *Research Directions in Concurrent Object Oriented Programming*, G. Agha, P. Wegner, A. Yonezawa (eds.), MIT Press, pp.257-280.
10. Andreoli, J-M. and Pareschi, R. 1991. "Linear Objects: Logical Processes with Built-in Inheritance", *New Generation Computing*, Vol. 9, Nos. 3-4, pp.445-473.
11. Anutariya, C. 1999. "A Declarative Description Data Model for XML Documents", Dissertation Proposal, CSIM, Asian Institute of Technology, Bangkok, Thailand, September.
12. Arocena, G.O., Mendelzon, A.O., Mihaila, G.A. 1997. "Applications of a Web Query Language", In *6th Int. WWW Conf.*, April, Available at `http://atlanta.cs.nchu.edu.tw/www/PAPER267.html`.
13. Aquilino, D., Asirelli, P., Formuso, A., Renso, C., and Turini, F. 2000. "Using MedLan to Integrate Geographical Data", In [104], pp.3-14.
14. Banatre, J.-P., and Le Matayer, D. 1993. "Programming by Multiset Transformations", *Communications of the ACM*, Vol. 36, No. 1, January, pp.98-111.
15. Banatre, J.-P., and Le Matayer, D. 1996. "Gamma and the Chemical Reaction Model: Ten Years After", In *Coordination Programming: Mechanisms, Models and Semantics*, J.-M. Andreoli, C. Hankin, and D. Le Metayer (eds.), Worl Scientific, pp.1-39.
16. BinProlog. 2000. "BinProlog Linda Library Manual", Available at `http://www.binnetcorp.com/BinProlog/index.html`.
17. Bonnet, Ph., Bressan, S., Leth, L., and Thomsen, B. 1996. "Towards ECLiPSe Agents on the Internet", In *Proc. of the 1st Workshop on Logic Programming Tools for Internet Applications*, P. Tarau, A. Davison, K. De Bosschere, and M. Hermenegildo (eds.), JICSLP'96, September, pp.1-9.
18. Brogi, A. and Ciancarini, P. 1991. "The Concurrent Language, Shared Prolog", *ACM Trans. on Programming Languages and Systems*, Vol. 13, No. 1, January, pp.99-123.
19. Bryce, C., Oriol, M., and Vitek, J. 1999. "A Coordination Model for Agents Based on Secure Spaces", In *Coordination Languages and Models: Coordination 99*, Amsterdam, April.

20. Bucci, A., Ciancarini, P., and Montangero, C. 1991. "Extended Shared Prolog: A Multiple Tuple Spaces Logic Language", In *Proc. 10th Japanese LP Conf.*, LNCS, Springer-Verlag.
21. Cabeza, D., Hermenegildo, M., and Varma, S. 1996. "The PiLLoW/CIAO Library for INTERNET/WWW Programming", In *Proc. of the 1st Workshop on Logic Programming Tools for Internet Applications*, P. Tarau, A. Davison, K. De Bosschere, and M. Hermenegildo (eds.), JICSLP'96, September, pp.43-62.
22. Calejo, M. 1999. "Java+Prolog: A Land of Opportunities", In *PACLP'99: The 1st Int. Conf. on Constraint Technologies and Logic Programming*, London, pp.1-2.
23. Carriero, N. and Gelernter, D. 1989. "Linda in Context', *Comms. of the ACM*, Vol. 32, No. 4, April, pp.444-458.
24. Carriero, N., Gelernter, D., and Zuck, L. 1996. "Bauhaus Linda", In *Object-based Models and Languages for Concurrent Systems*, P. Ciancarini, O. Nierstrasz, A. Yonezawa (eds.), LNCS 924, Springer-Verlag, July, pp.66-76.
25. Cardelli, L. 1999. "Abstractions for Mobile Computation", In *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, J. Vitek and C. Jensen (eds.), LNCS 1603, Springer-Verlag, pp.51-94.
26. Cardelli, L. and Gordon, A.D. 1998. "Mobile Ambients", In *FoSSaCS'98*, LNCS 1378, Springer-Verlag, pp.140-155.
27. Chandy, K.M. and Kesselman, C. 1993. "CC++: A Declarative Concurrent Object-oriented Programming Notation", In *Research Directions in Concurrent Object Oriented Programming*, MIT Press.
28. Chu, D. and Clark, K.L. 1993. "IC-Prolog II: A Multi-threaded Prolog System", In *Proc. of the ICLP'93 Post Conf. Workshop on Concurrent, Distributed and Parallel Implementations of LP Systems*, Budapest.
29. Ciancarini, P. 1991. "Parallel Logic Programming using the Linda Model of Computation", In *Research Directions in High-level Parallel Programming Languages*, J.P. Banatre and D. Le Metayer (eds.), LNCS 574, June, pp.110-125.
30. Ciancarini, P. 1994. "Distributed Programming with Logic Tuple Spaces", *New Generation Computing*, Vol. 12, No. 3, May, pp.251-284.
31. Ciancarini, P., Knoche, A., Tolksdorf, R., Vitali, F. 1996b. "PageSpace: An Architecture to Coordinate Distributed Applications on the Web", *Computer Networks and ISDN Systems*, Vol. 28, pp.941-952.
32. Ciancarini, P., Mazza, M., and Pazzaglia, L. 1998. "A Logic for a Coordination Model with Multiple Spaces", *Science of Computer Programming*, Vol. 31, Nos. 2-3, pp.231-262.
33. Ciancarini, P. and Rossi, D. 1998. "Coordinating Java Agents over the WWW", *World Wide Web Journal*, Vol. 1, No. 2, pp.87-99.
34. Ciancarini, P., Omicini, A., and Zambonelli, F. 1999. "Coordination Technologies for Internet Agents", *Nordic Journal of Computing*, Vol. 6, pp.215-240.
35. Clark, K.L. and Gregory, S. 1986. "Parlog: Parallel Programming in Logic", *ACM Trans. Programming Languages and Systems*, Vol. 8, No. 1, pp.1-49.
36. Clark, K.L. and McCabe, F.G. 2000. "Go! – A Logic Programming Language for the Internet", *DRAFT*, July.
37. Clark, K.L. and Robinson, P.J. 2000. "Agents as Multi-threaded Logical Objects", November. Available at `http://www-lp.doc.ic.ac.uk/~klc/qp3.html`.
38. Clark, K.L., Robinson, P.J., and Hagen, R. 1998. "Programming Internet Based DAI Applications in Qu-Prolog", In *Multi-agent Systems*, LNAI 1544, Springer-Verlag.
39. Coulouris, G., Dollimore, J., and Kindberg, T. 1994. *Distributed Systems: Concepts and Design*, Addison-Wesley, 2nd ed.

40. Craven, M., DiPasquo, D., Freitag, D., McCallum, A., Mitchell, T., Nigam, K., and Slattery, S. 1998. "Learning to Extract Symbolic Knowledge from the World Wide Web", In *Proc. of the 15th Nat. Conf. on AI (AAAI-98)*.
41. Cremonini, M., Ominici, A., and Zambonelli, F. 1999. "Multi-agent Systems on the Internet: Extending the Scope of Coordination towards Security and Topology", In *Proc. of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAMAAW'99)*, LNAI 1647, Springer-Verlag, pp.77-88.
42. Cremonini, M., Ominici, A., and Zambonelli, F. 2000. "Ruling Agent Motion in Structured Environments", In *HPCN Europe 2000*, LNCS, May.
43. Cunha, J.C. and Marques, R.F.P. 1996. "PVM-Prolog: A Prolog Interface to PVM", In *Proc. of the 1st Int. Austrian-Hungarian Workshop on Distributed and Parallel Systems, DAPSYS'96*, Miskolc, Hungary.
44. Davison, A. 1993. "A Survey of Logic Programming-based Object Oriented Languages", In *Research Directions in Concurrent Object Oriented Programming*, G. Agha, P. Wegner, A. Yonezawa (eds.), MIT Press.
45. Davison, A. 1999. "A Concurrent Logic Programming Model of the Web", In *PA-CLP'99: The 1st Int. Conf. on Constraint Technologies and Logic Programming*, London, pp.437-451.
46. De Bosschere, K. and Jacquet, J.-M. 1992. "Comparative Semantics of μlog", In *Proc. of the PARLE'92 Conf.*, D. Etiemble and J.-C. Syre (eds.), LNCS 605, Springer-Verlag, pp.911-926.
47. De Bosschere, K. and Jacquet, J.-M. 1996a. "μ^2log: Towards Remote Coordination", In *1st Int. Conf. on Coordination Models, Languages and Applications (Coordination'96)*, P. Ciancarini and C. Hankin (eds.), Cesena, Italy, LNCS 1061, Springer-Verlag, April, pp.142-159.
48. De Bosschere, K. and Jacquet, J.-M. 1996b. "Extending the μlog Framework with Local and Conditional Blackboard Operations", *Journal of Symbolic Computation*.
49. De Bosschere, K. and Tarau, P. 1996. "Blackboard Extensions in Prolog", *Software–Practice and Experience*, Vol. 26, No. 1, January, pp.49-69.
50. Decker, S., Brickley, D., Saarela, J., and Angele, J. 1998. "A Query and Inference Service for RDF", In *W3C Workshop on Query Languages for XML*, Available at `http://www.w3.org/TandS/QL/QL98/pp/queryservice.html`
51. Denti, E., Natali, A., and Omicini, A. 1998. "On the Expressive Power of a Language for Programming Coordination Media", In *Proc. of the 1998 ACM Symp. on Applied Computing (SAC'98)*, February, pp.167-177.
52. Diaz, M., Rubio, B., and Troya, J.M. 1996. "Distributed Programming with a Logic Channel-based Coordination Model", *The Computer Journal*, Vol. 39, No. 10, pp.876-889.
53. Diaz, M., Rubio, B., and Troya, J.M. 1997a. "DRL: A Distributed Real-time Logic Language", *Computer Languages*, Vol. 23, Vol. 2-4, pp.87-120.
54. Elshiewy, N.A. 1990. *Robust Coordinated Reactive Computing in Sandra*, Thesis for Doctor of Technology, Royal Institute of Technology KTH and Swedish Institute of Computer Science (SICS), RIT (KTH) TRITA-TCS-9005, SICS/D-90-9003.
55. Etzioni, O. and Weld, D.S. 1995. "Intelligent Agents on the Internet: Fact, Fiction, and Forecast", *IEEE Expert*, pp.44-49, August.
56. Fernandez, M., Simeon, J., and Wadler, P. (eds.). 2000b. "XML Query Languages: Experiences and Exemplars", Available at `http://www-db.research.bell-labs.com/user/simeon/xquery.html`

57. Foster, I. 1992. "Information Hiding in Parallel Programs", Tech. Report MCS-P290-0292, Argonne National Lab.
58. Foster, I. and Taylor, S. 1989. *Strand: New Concepts in Parallel Programming*, Prentice Hall.
59. Foster, I. and Taylor, S. 1992. "A Compiler Approach to Scalable Concurrent Program Design", Tech. Report MCS-P306-0492, Argonne National Lab.
60. Franklin, S. and Graesser, A. 1996. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents", In *Proc. of the 3rd Int. Workshop on Agent Theories, Architectures and Languages*, Springer Verlag. Available from `http://www.msci.memphis.edu/~franklin/AgentProg.html`
61. Geist, G.A. Beguelin, A., Donjarra, J., Jiang, W., Manchek, R., and Sunderam, V. 1994. *PVM: Parallel Virtual Machine A Users' Guide and Tutorial for Networked Parallel Computing*, MIT Press. Available at `http://www.netlib.org/pvm3/book/pvm-book.html`.
62. Geist, G.A., Kohl, J.A., and Papadopoulos, P.M. 1996. "PVM and MPI: A Comparison of Features", *Calculateurs Paralleles*, Vol. 8, No. 2.
63. Gelernter, D. 1989. "Multiple Tuple Spaces in Linda", In *Proc. PARLE'89: Parallel Architectures and Languages Europe*, June, pp.20-27.
64. Goh, C.H., Bressan, S., Madnick, S., and Siegel, M. 1999. "Context Interchange: New Features and Formalisms for the Intelligent Integration of Information", *ACM Trans. on Info. Systems*, Vol. 17, No. 3, July, pp.270-293.
65. Haridi, S., Van Roy, P., Brand, P., Mehl, M., Scheidhauer, R., and Smolka, G. 1999. "Efficient Logic Variables for Distributed Computing", *ACM Trans. in Programming Languages and Systems*, Vol. 21, No. 3, May, pp.569-626.
66. Haridi, S., Van Roy, P., Brand, P., and Schulte, C. 1998. "Programming Languages for Distributed Applications", *New Generation Computing*, Vol. 16, No. 3, May, pp.223-261.
67. Hewitt, C. 1985. "The Challenge of Open Systems", *Byte*, April, pp.223-233.
68. Hirsch, M., Silverman, W., and Shapiro, E.Y. 1987. "Computation Control and Protection in the Logix System", In *Concurrent Prolog: Collected Papers, Vol. 2*, E.Y. Shapiro (ed.), MIT Press, Chapter 20, pp.28-45.
69. Hong, T.W. and Clark, K.L. 2000. "Concurrent Programming on the Web with Webstream", August. Avalable at: `http://longitude.doc.ic.ac.uk/~twh1/longitude/`.
70. Hosoya, H. and Pierce, B.C. 2000. "XDuce: A Typed XML Processing Language", Preliminary Report, Dept. of CIS, Univ. of Pennsylvania, March 8th.
71. Hull, R. 1997. "Managing Semantic Heterogeneity in Databases: A Theoretical Perspective", Invited tutorial at *16th ACM Symp. on Principles of Databases Systems (PODS'97)*, Available at `http://www-db.research.bell-labs.com/user/hull/pods97-tutorial.html`.
72. I3 Program. 1995. *Reference Architecture for the Intelligent Integration of Information*, Version 2.0 (draft), DARPA. Available at `http://dc.isx.com/I3/html/briefs/I3brief.html`.
73. Jacquet, J.-M. and De Bosscher, K. 1994. "On the Semantics of μlog", *Future Generation Computer Systems*, Vol. 10, pp.93-135.
74. Jeffery, D., Dowd, T., and Somogyi, Z. 1999. "MCORBA: A CORBA Binding for Mercury", Tech. Report., Dept. of CSSE, Univ. of Melbourne, Australia.
75. Kahn, K.M., and Miller, M.S. 1988. "Language Design and Open Systems", In *The Ecology of Computation*, B. Huberman (ed.), North Holland.

76. Klinger, S., Yardeni, E., Kahn, K., and Shapiro, E.Y. 1988. The Language FCP(:,?), In *Proc. of the Int. Conf. on 5th Generation Computer Systems*, pp.763-773.
77. Kowalski, R. 1985. "Logic-based Open Systems", Technical Report, Dept. of Computing, Imperial College, September.
78. Kowalski, K., and Sadri, F. 1996. "Towards a Unified Agent Architecture that Combines Rationality with Reactivity", In *Proc. of Int. Workshop on Logic in Databases*, San Miniato, Italy, Springer-Verlag.
79. Kowalski, K., and Sadri, F. 1999. "From Logic Programming to Multi-Agent Systems", *Annals of Mathematics and Artificial Intelligence*, Vol. 25, pp.391-419.
80. Lakshmanan, L.V.S., Sadri, F., and Subramanian, I.N. 1996. "A Declarative Approach to Querying and Restructuring the World-Wide-Web", In *Post-ICDE Workshop on Research Issues in Data Engineering (RIDE'96)*, New Orleans, February. Available as `ftp://ftp.cs.concordia.ca/pub/laks/papers/ride96.ps.gz`.
81. Levy, A.Y., Rajaraman, A., and Ordille, J.J. 1996. "Querying Heterogeneous Information Sources using Source Descriptions", In *Proc. of the Int. Conf. on Very Large Databases*, September.
82. Liu, M. 1999. "Deductive Database Languages: Problems and Solutions", *ACM Computing Surveys*, Vol. 31, No. 1, March, pp.27-62.
83. Loke, S.W. and Davison, A. 1998. "LogicWeb: Enhancing the Web with Logic Programming", *Journal of Logic Programming*, 36, pp.195-240.
84. Loke, S.W. and Davison, A. 2001. "Secure Prolog Based Mobile Code", *Theory and Practice of Logic Programming*, Vol. 1, No. 1, To Appear.
85. LPA 1997. `LPA ProWeb Server`. Available at `http://www.lpa.co.uk/`.
86. Ludascher, B., Himmeroder, R., Lausen, G., May, W., and Schlepphorst, C. 1998. "Managing Semistructured Data with FLORID: A Deductive Object-Oriented Perspective", *Information Systems*, Vol. 23, No. 8, pp.1-25.
87. Marchiori, M. and Saarela, J. 1998. "Query + Metadata + Logic = Metalog", In *W3C Workshop on Query Languages for XML*, Available at `http://www.w3.org/TandS/QL/QL98/pp/metalog.html`
88. McCabe, F.G. and Clark, K.L. 1995. "April: Agent PRocess Interaction Language", In *Intelligent Agents: Theories, Architectures, and Languages*, M. Wooldridge, N.R. Jennings (eds.), LNAI 890, Springer-Verlag, pp.324-340.
89. Mehl, M., Schulte, C., and Smolka, G. 1998. "Futures and By-need Synchronization for Oz", Tech. Report, Programming Systems Lab, DFKI and Univ. des Saarlandes, May.
90. Meijer, E. and Sheilds, M. 2000. "XMLambda: A Functional Language for Constructing and Manipulating XML Documents", Tech Report, Oregon Graduate School.
91. Mendelzon, A.O. and Milo, T. 1997. "Formal Models of Web Queries", In *Proc. of the ACM SIGACT-SIGMOD-SIGART Symp. on Principles of Database Systems (PODS)*, May, pp.134-143.
92. Menezes, R., Merrick, I., and Wood, I. 1999. "Coordination in a Content-Addressable Web", *Autonomous Agents and Multi-Agent Systems*, Vol. 2, pp.287-301.
93. Meng, W. and Yu, C. 1995. *Query Processing in Multidatabase Systems*, Addison-Wesley.
94. Miller, M.S., Bobrow, D.G., Tribble, E.D., and Levy, J. 1987. "Logical Secrets", In *Logic Programming: Proc. 4th Int. Conf. (ICLP'87)*, pp.704-728.

95. Moore, J.T. 1998. "Mobile Code Security Techniques", CIS, Univ. of Pennsylvania, May.
96. Omicini, A. 1999. "On the Semantics of Tuple-based Coordination Models", In *Proc. of the 1999 ACM Symp. on Applied Computing (SAC'99)*, February, pp.175-182.
97. Omicini, A., Denti, E., and Natali, A. 1995. "Agent Coordination and Control through Logic Theories", AI*IA'95: In *Proc. of the 4th AI*IA Congress*, LNAI 992, Springer-Verlag, pp.439-450.
98. Papadopoulos, G.A. and Arbab, F. 1998. "Coordination Models and Languages", In *Advances in Computers*, Vol. 46, Academic Press, pp.329-400.
99. Pappas, A. (coordinator) 1999. *Workshop on Logic Programming and Distributed Knowledge Management*, ALP-UK/ALP/Compulog-net Tutorial/Workshop at PA Expo99, London, April.
100. Papakonstantinou, Y., Abiteboul, S., and Garcia-Molina, H. 1996. "Object Fusion in Mediator Systems", In *Proc. of the Int. Conf. on Very Large Databases*, September.
101. Pontelli, E. and Gupta, G. 1997. "W-ACE: A Logic Language for Intelligent Internet Programming", ICTAI'97, In *Proc. of the IEEE 9th Int. Conf. on Tools with AI*, pp.2-10. A much expanded version of this paper can be found at `http://www.cs.nmsu.edu/lldap/pri_lp/web/`.
102. Pontelli, E. and Gupta, G. 1998. "WEB-KLIC: A Concurrent Logic-based Unified Framework for Internet Programming", *AITEC Contract Research Projects in FY 1998: Proposal*, Tokyo, Japan, Available at `http://icot10.icot.or.jp/AITEC/FGCS/funding/98/plan07.html`.
103. Ramakrishnan, R. and Silberschatz, A. 1998. "Scalable Integration of Data Collections on the Web", Tech. Report, Univ. of Wiscosin-Madison.
104. Ramakrishnan, R. and Subrahmanian, V.S. (guest editors) 2000. *Journal of Logic Programming Special Issue on Logic-based Heterogeneous Information Systems*, Vol. 43, No. 1, April.
105. Saraswat, V.J. 1993. *Concurrent Constraint Programming*, MIT Press.
106. Saraswat, V.M., Kahn, K., and Levy, J. 1990. "Janus: A Step towards Distributed Constraint Programming", In *North American Conf. on LP*, MIT Press, October, pp.431-446.
107. Scroeder, M., Marques, R., Wagner, G., and Cunha, J.C. 1997. "CAP – Concurrent Action and Planning: Using PVM-Prolog to Implement Vivid Agents", In *PAP'97: The 5th Int. Conf. on the Practical Applications of Prolog*, London, pp.271-289.
108. Shanmugasundaram, J., Tufte, K., He, G., Zhang, C., DeWitt, D., and Naughton, J. 1999. "Relational Databases for Querying XML Documents: Limitations and Opportunities", In *Proc. of the 25th VLDB Conf.*, Edinburgh, Scotland.
109. Shapiro, E.Y. 1989a. "The Family of Concurrent Logic Programming Languages", *ACM Computing Surveys*, Vol. 21, No. 3, September, pp.413-510.
110. SICS. 1999 "SICStus Prolog Manual", Swedish Institute of Computer Science, Kista, Sweden, Available at `http://www.sics.se/sicstus/`.
111. Slattery, C. and Craven, M. 1998. "Combining Statistical and Relational Methods for Learning in Hypertext Domains", In *Proc. of the 8th Int. Conf. on Inductive LP*.
112. Sundstrom, A. 1998. *Comparative Study between Oz 3 and Java*, Master's Thesis 1998-10-01, CS Dept., Uppsala University, Sweden.
113. Sun Microsystems, 1999. "The JavaSpaces Specification", Available at `http://java.sun.com/docs/books/jini/javaspaces`.

114. Sutcliffe, G. and Pinakis, J. 1992. "Prolog-D-Linda", Technical Report 91/7, Dept. of CS, Univ. of Western Australia, Western Australia.
115. Szeredi, P., Molnár, K., and Scott, R. 1996. "Serving Multiple HTML Clients from a Prolog Application", In *Proc. of the 1st Workshop on Logic Programming Tools for Internet Applications*, P. Tarau, A. Davison, K. De Bosschere, and M. Hermenegildo (eds.), JICSLP'96, September, pp.81-90.
116. Taguchi, K., Sato, H., and Araki, K. 1998. "TeleLog: A Mobile Logic Programming Language", Tech. Report, Chikushi Jyogakuen Univ. Japan.
117. Tarau, P. 1999. "Jinni: Intelligent Mobile Agent Programming at the Intersection of Java and Prolog", *Proc. of PAAM'99*, London.
118. Tarau, P. and Dahl, V. 1998. "A Coordination Logic for Agent Programming in Virtual Worlds", In *Coordination Technology for Collaborative Applications - Organizations, Processes, and Agents*, W. Conen and G. Neumann (eds.), Springer-Verlag.
119. Tarau, P. and Dahl, V. 1998. "Mobile Threads through First Order Continuations", In *Proc. of APPAI-GULP-PRODE'98*, Coruna, Spain, July.
120. Tarau, P., De Bosschere, K, Dahl, V., and Rochefort, S. 1999. "LogiMOO: An Extensible Multi-User Virtual World with Natural Language Control", *Journal of Logic Programming*, Vol. 38, No. 3, March, pp.331-353.
121. Taylor, S., Av-Ron, E., and Shapiro, E.Y. 1987. "A Layered Method for Process and Code Mapping", *New Generation Computing*, Vol. 5, No. 2.
122. Tick, E. 1995. "The Deevolution of Concurrent Logic Programming Languages", *Journal of Logic Programming*, 10th Anniversary Special Issue, Vol. 23, No. 2, p.89-123.
123. Thorn, T. 1997. "Programming Languages for Mobile Code", *ACM Computing Surveys*, Vol. 29, No. 3, pp.213-239, September.
124. Van Roy, P. 1999. "On the Separation of Concerns in Distributed Programming: Application to Distribution Structure and fault Tolerance in Mozart", In *Int. Workshop on Parallel and Distributed Computing for Symbolic and Irregular Applications (PDSIA'99)*, July.
125. Van Roy, P. and Haridi, S. 1999. "Mozart: A Programming System for Agent Applications", In *Int. Conf on Logic Programming (ICLP'99), Int. Workshop on Distributed and Internet Programming with Logic and Constraint Languages*, November.
126. Van Roy, P., Haridi, S., Brand, P., Smolka, G., Mehl, M., and Scheidhauer, R. 1997. "Mobile Objects in Distributed Oz", *ACM Trans. on Programming Languages and Systems*, Vol. 19, No. 5, September, pp.804-851.
127. Wadler, P. 2000. "XML: Some Hyperlinks Minus the Hype", Bell Labs, Lucent Technologies, Available at `http://cm.bell-labs.com/cm/cs/who/wadler/xml/`.
128. Waldo, J., Wyant, G., Wollrath, A., and Kendall, S. 1994. "A Note on Distributed Computing", Tech Report SMLI TR-94-29, Sun Microsystems Lab, November.
129. Wetzel, G., Kowalski, R.A., and Toni, F. 1995. "A Theorem Proving Approach to CLP", In *Proc. of the 11th Workshop on LP*.
130. W3C. 1998. *Extensible Markup Language (XML)*, W3C Recommendation, 10th February, REC-xml-19980210 Available at `http://www.w3.org/TR/REC-xml`.
131. W3C 2000b. *Resource Description Framework (RDF)*, 5th May, Available at `http://www.w3.org/RDF/`.