# ERC Advanced Grant 2014

## Research proposal [Part B2]

## B2: The scientific proposal

### Our objective

The objective of the $\mathcal{AWOL}$ project is the design, implementation, experimentation, and demonstration of a net-aware knowledge-representation and automated-reasoning system offering a radical alternative to mainstream Semantic Web (SW) technology for an $\mathcal{A}$lternative $\mathcal{W}$eb $\mathcal{O}$ntology $\mathcal{L}$anguage—with, at its vehicle, the $\mathcal{HOOT}$ "Ontological Programming" language.[1] The key to this objective's scientific locks—scalability and distribution of SW processing—is the "*Order-Sorted Feature*" ($\mathcal{OSF}$) constraint formalism, as well as its relevant implementation technology influenced by the $\mathcal{LIFE}$ Constraint Logic Programming language in the context of today's Internet. $\mathcal{HOOT}$ is to provide an operational system that:[2]

- uses $\mathcal{OSF}$ terms, which are syntactic notations for labelled graph structures, as universal representation;[3]
- combines computational semantics from Functional Programming (FP), Constraint Logic Programming (CLP), and Object-oriented Progrraming (OOP) through the formal view of objects as $\mathcal{OSF}$ graphs, and classes as efficiently enforceable functionally and relationally constrained object-structure specifications;
- defines functions as curryable rules over $\mathcal{OSF}$ terms, and relations as constrained Horn clauses over $\mathcal{OSF}$ terms; and,
- uses such a formalism for specifying (and verifying operationally) consistent object structures that are distributed over the Internet that must verify constraints from their class definitions as well as those inherited from superclasses.

In the course of the $\mathcal{CEDAR}$ project, we initiated work on testing the pragmatics of existing tools and techniques that have been proposed for the Semantic Web. Among the most important properties is their ability (or lack thereof) to reason effectively over knowledge bases and deal with enormous quantities of distributed data governed by this distributed knowledge. $\mathcal{CEDAR}$'s key outcome has been a solid understanding and hands-on experience with Semantic Web technologies, as well as with Big Linked Data technology.[4]

I have had a well-documented public stance on how the $\mathcal{OSF}$ unification-based approach I developed offers a viable alternative to the official DL-based Semantic Web languages [4, 5]. The reason is that it is formally simpler, usably more intuitive to the designer, and practically more efficient. My scientific conviction has been recently reinforced with the practical performance experiments carried out since 2013 as part of the $\mathcal{CEDAR}$ project. Hence, the $\mathcal{AWOL}$ project proposal's rationale is to leverage well-understood and well-honed architectures and algorithms from (Concurrent) Constraint (Logic) Programming [33, 34, 3] and Big (Linked) Data [13, 26] in the specific area of ontological reasoning [20]. I have explicated in detail in my previous research how Prolog's efficient implementation techniques (using the Warren Abstract Machine design [1]) can be adapted to compile $\mathcal{OSF}$ unification [12], even modulo relationally and functionally constrained concept definitions [22].

However, the leap forward from $\mathcal{LIFE}$ to making the new proposed ontological programming $\mathcal{HOOT}$ an alternative language to OWL for the $\mathcal{AWOL}$ project is that $\mathcal{HOOT}$:

- refers to objects distributed all over the Internet (and not just in a local RAM or silo-ed DB);
- represents all $\mathcal{OSF}$ graphs as RDF(S) KB/DBs;[5] and,

[1] $\mathcal{HOOT}$ stands for $\mathcal{H}$ierarchical $\mathcal{O}$ntologies with $\mathcal{O}$bjects and $\mathcal{T}$ypes.

[2] We shall use the words "*class*," "*sort*," and "*concept*" interchangeably as symbols denoting sets of instances. Similarly "*subclass*," "*subsort*," and "*subconcept*" (or "*is-a*"), all denote set inclusion. These different terminologies come from programming languages, logic, and knowledge representation, respectively.

[3] The use of such a universal notation for representing code syntax and data structure alike eases meta-programming through a quote/unquote meta-operator, just as S-Expressions in LISP, or Prolog terms.

[4] http://cedar.liris.cnrs.fr/

[5] So-called TBoxes/ABoxes in the new Semantic Web lingo. A "TBox" (for "Terminological Box") is a set of (DL) axioms constraining data contained in an "Assertional Box" or "ABox." The TBox is the Knowledge Base (KB) and the ABox is the Data Base (DB). Equivalent names for Tbox/Abox in more conventional terminology are Schema/Database (also Intensional/Extensional DB).

- performs reasoning with *lazy memoizing* to prevent uselessly repeated computations as well as to handle potential undecidability of some constraints by sacrificing completeness for convergence.[6]

The expected outcome of $\mathcal{AWOL}$ is to demonstrate beyond tenable doubt that $\mathcal{OSF}$ constraint logic can be the basis of scientifically formal, operationally effective, and practically efficient "Ontological Programming" over distributed RDF-based knowledge and data bases.

## The State of the Art

### Semantic Web Reasoners

In here give a brief description of the most prominent extant SW reasoners. We limit our selection to systems that are full-fledged *reasoners*, and not just *classifiers*. This is because our interest goes beyond concept classification and includes Boolean query answering as well. This rules out systems such as ELK [36], CEL [25], CB [35], *etc.*, that do not support query answering.

We have chosen as a representative set the following system SW reasoners: FaCT++; HermiT; Pellet; TrOWL; Racerpro; and, SnoRocket.

FaCT++ (Fast Classification of Terminologies) is a reasoner developed at the University of Manchester [49]. It is based on the Description Logic fragment $\mathcal{SHOIQ}$ [32]. It is implemented in C++ as a deductive tableau [39] adapted to the specifics of this logic. It is claimed to use a wide range of heuristic optimizations. FaCT++ provides TBox reasoning (subsumption, satisfiability, classification) and partial support for ABox processing (retrieval).

HermiT is also a reasoner for a (slight extension) of the Description Logic fragment $\mathcal{SHOIQ}$.[7] It is based upon hypertableau reasoning, an optimized version of tableau reasoning [40]. It purports to provide a faster process for classifying ontologies. The main optimization of hypertableau *vs.* tableau that it tries to minimize nondeterminism in the treatment of disjunctions and is more memory-efficient. HermiT provides TBox reasoning, with the ability of checking the consistency of an ontology and inferring implicit relationships between concepts.

Pellet is a free open-source Java-based reasoner [45]. It, too, is based on the tableau algorithm and supports the Description Logic fragment $\mathcal{SHOIN}(D)$. It provides TBox reasoning (subsumption, satisfiability, and classification) and ABox reasoning (retrieval, conjunctive query answering). It uses many optimization techniques and supports entailment checks and ABox querying through its interface.

TrOWL (Tractable reasoning infrastructure for OWL 2) was developed at the University of Aberdeen [48]. This is a system that starts by transforming an ontology from OWL-DL to OWL-QL [28] in order to classify it in polynomial time. Under this transformation, conjunctive query answering and consistency checking remain the same as for OWL-DL. In addition, TrOWL can generate a database schema for storing normalized representations of OWL-QL ontologies.

Racerpro is a commercial version of RACER (Renamed ABoxes and Concept Expression Reasoner) [31, 30]. It implements a reasoner for the description logic $\mathcal{SHIQ}$. RACER provides both TBox and ABox reasoning. It supports all the optimizations of FaCT++ as well as new techniques for dealing with number restrictions and ABoxes.

Snorocket [37] was proposed as a high-performance implementation of a polynomial-time classification algorithm for the lightweight Description Logic $\mathcal{EL}$ [24].[8] It was primarily meant to be optimized for classifying SNOMED CT. It can process only conjunctive queries.

**Critical analysis** The main criticism this PI holds toward these DL-based languages is their inability to scale up to very large ontologies. One of the objectives of the $\mathcal{CEDAR}$ project was to test the foregoing reasoners on this aspect. We benchmarked them, along our own proof-of-concept prototype, on publicly available ontologies of

[6] This lazy semantics, incidentally, adheres effectively to a *final*—as opposed to *initial*—algebra interpretation of $\mathcal{OSF}$ terms (which can form infinitely descending approximation chains [2]).

[7] This fragment is called $\mathcal{SHOIQ}^+$ [44].

[8] Description Logics in the $\mathcal{EL}$-family are weaker versions that provide existential roles ($\exists r.C$) but no universal roles ($\forall r.C$). This amounts to being able to specify taxonomies with domain/range constraints for roles and attributes (*i.e.*, functional features and set-valued functional features as done in [9]).

enormous sizes: Wikipedia,[9] BioModels,[10] MeSH,[11] and NCBI.[12] Focusing first on basic taxonomic reasoning (*i.e.*, involving only concepts but no attributes), the results showed that, for the same queries, our prototype was several orders of magnitude more efficient than all these DL-based reasoners; or rather, than those of them that could return at all [10, 23]. Proceeding with the same experiments on attributed concept taxonomies led to similar results [11].

Another, no less important, criticism this PI shares with several actual users of OWL, is its being unintuitive. The first obstacle to a user is its undue complication—both at the syntax and semantic level. One of the main drawbacks experienced by field workers using OWL technology for Ontological Programming is its insistance on so-called "Open-World" (OW) semantics (opposed to "Closed-World" (CW) semantics, by which LP, OOP, and database systems abide). Basically, OW semantics makes the assumption that, in a complete proof system, a logical sentence will be deemed false only if it can be so proven explicitly. Whereas, CW semantics considers false any sentence that cannot be proven true even when the proof system's reasoning capabilities are limited (*i.e.*, polymorphic type inference), or its proof strategy is incomplete (*e.g.*, Prolog's left-right/depth-first Horn-clause resolution). In the PI's experience, OW is actually useful in much rarer use cases and is unfortunately at the origin of much voiced frustration (see, *e.g.*, [47, 29]).

Still, large efforts have been invested in using OWL as the representation and reasoning language for very large ontologies (such as MeSH or NCBI cited above). It is however not so clear *what* these knowledge bases are actually used for. If one is to use OWL only as a formal notation, then such languages as Z, OCL, or even simply UML, are much more familiar to, and *actually used* by, real users.

Knowledge-representation systems using an OW assumption prevent using implicit knowledge (such as default conclusions) on the grounds that it would render the logic non-monotonic. In effect, such a system must then have three truth values: "*true*," "*false*,", and "*undecided*." Thus, this makes dealing with undecidable but recursively enumerable decisions impossible (*e.g.*, Horn Logic). Besides, actual knowledge evolves non-monotonically in real life; so, having a non-monotomic inference system is not a hinder to its use. What is problematic, on the other hand, is that the OW assumption is counter-intuitive to many users in addition to being impractical since it requires adding an undue amount of specific information to cover anything known to be false.[13]

On the other hand, the CW assumption takes pragmatic computational advantage of default knowledge while relieving knowledge-base developers from having to specify as well properties known not to hold. Finally, if one actually *needs* OW reasoning on some specific knowledge, it is always possible to add to such axioms continuations in the style of $\mathcal{LIFE}$'s residuation [17, 16]—*i.e.*, a suspended computation waiting for more information (a kind of conditional truth: "*true, if a few pending constraints are verified*") [19].

### The Resource Description Framework

Clearly, as the foregoing statements show, I believe that the W3C went astray with putting all its Semantic Web eggs the single OWL basket of DL-based technology. On the other hand, I also believe that it has been quite judicious in proposing the Resource Description Framework (RDF) as a more elementary and less pretentious *lingua franca* for a universal representation of all Semantic Web information. In this regard, it has acknowledged the fact that all such information is representable as a set of web-wide interconnected labelled graphs—a view that I fully espouse. RDF is indeed a simple notation for such objects where *everything* is denoted by a *Uniform Resource Identifier* (URI). Most importantly, no operational semantics, let alone formal reasoning, has been ascribed to it. It is just a *notational system*—and that is that.

This, in my opinion, is as clever as it is simple an idea in that it sets an agreed upon standard for sharing distributed information. This is because it does not presume any *a priori* semantics other than being a conventional notation for distributed labelled graphs.

[9] This is an ontology derived from the Wikipedia online database and containing 111,599 concepts.

[10] This is an ontology of various biological models containing 182,651 concepts.

[11] This is an ontology of Medical Subject Headings of the National Library of Medicine; it contains 286,381 concepts.

[12] This is the National Center for Biotechnology Information's ontology of all known living organisms; it contains 903,617 concepts.

[13] For example, this document is a real-life OWL use case report [29]. Curiously, it is one of the use cases cited in the official OWL 2 documentation meant to illustrate the usefulness of OWL as an ontology specification and reasoning system "*in the field*." Unless the OWL 2 document's authors cited it without actually reading it, it is clearly a *de facto* admission on their part of the counter-intuitiveness and ineffectiveness (not to mention intolerable inefficieny) of the system they expect "Mr. User" to choose for ontological reasoning.

Further elaborations of RDF, notably RDF Schema (RDFS), RDF with attributes (RDFa), and the Simple Knowledge Organization System (SKOS), are also RDF-based notational systems proposing purely representational conventions for (respectively) typed, attributed, and taxonomically organized graph-based information. As such, they do not have the pretense of being reasoning systems. Even as "knowledge-representational" systems, they merely offer RDF-based notational conventions for oft-used constructs in knowledge representation. No *a priori* formal proof-theoretic interpretation is imposed: only informal, though familiar, syntactic constructs are made available.

RDF comes with an XML-based surface syntax, but its internal data format is that of a set of triples (graph edges of the form $\langle subject, predicate, object \rangle$) stored in so-called *triplestores*. There is no name scoping in RDF.[14]

There are alternative more convenient (for humans) shorthand notations for RDF triples other than its verbose and awkward XML syntax (*e.g.*, N3, or Turtle). One in particular is growing in popularity: the Java Script Object Notation (JSON). It has become a *de facto* standard as "*a lightweight data-interchange format [. . . ] easy for humans to read and write, [. . . and] for machines to parse and generate.*" JSON describes an object (*i.e.*, a graph) as a set of Key/Value pairs (two double-quoted strings). It has emerged out of the JavaScript programming community as a standard data representation. It maps to RDF (and vice-versa)—up to shared conventions. It may be viewed as a more user-friendly (*i.e.*, readable) as well as more economical (trees *vs.* sets of edges) notation. It has however its own conventions.

"JSON Linked Data" (JSON-LD) adds to JSON the notion of "context." Such a context is a kind of type "ontology," a schema constraining the nature and structure of JSON data referring to it. All objects must abide by the definitional structure edicted for them by the schema they refer to. For this, JSON-LD uses a standard "normal form" obtained using so-called "*format algorithms*" (conventional syntax normalization) for JSON-LD.[15]

For a comprehensive overview of basic informal notational conventions for expressing "knowledge" representations using RDF, see this presentation. As well, if only to get an idea of the current lack of general agreement in the design and use of RDF-derived dialects, it is worth mentioning this "RDF-bashing" set of comments by one of its designers.[16]

In terms of implementation resources and tools, one of the most reliable and open-source system for a wide variety of Semantic Web processing services is indubitably Jena.[17] It is uncommitted as to which system to recommend for the Semantic Web. It merely offers well-designed software-engineering modules implementing various specifications issued from various Semantic-Web W3C Working Groups. It offers an impressive set of system-level and reasoning-level primitives. It is Java-based (and thus Scala-compatible).[18]

### Alternative approaches: Constraint-based distributed computing

As seen in the previous overview, the vast majority of ontology specification systems and languages today are simple notational conventions carrying no formal proof-theoretic semantics. The only ones that come as well with a formal operational semantics for reasoning are based on Description Logic, the foremost representative of which is the Web Ontology Language (OWL), an ontology specification and reasoning formalism adopted as a recommendation by the W3C. This, however, has proven to yield more than questionable usefulness [47], let alone performance or scalability [4].

An alternative that has curiously been largely ignored, arises from (Constraint) Logic Programming. However, despite its relative loss of public interest since its heydays in the 1980s, Logic Programming technology still somehow renders impressive services. This is illustrated for example by the specification and processing of Natural Language Processing and knowledge-based reasoning needed to make out a meaning expressed for human agents. Such use of Prolog in IBM Watson's Jeopardy Win has been acknowledged among the critical strengths of this software's "reasoning" power.[19,20] This is not a simple reasoning feat, and indeed quite impressive. It is this sort of Prolog technology, improved with constraint-solving, functional programming, and recent advances

[14] http://manu.sporny.org/2013/rdf-identifiers/
[15] http://www.w3.org/TR/json-ld-api/
[16] Warning—crude opinionated language!
[17] https://jena.apache.org/
[18] As justified later, Scala will be the $\mathcal{AWOL}$ project's implementation language choice.
[19] http://www.cs.nmsu.edu/ALP/wp-content/uploads/2011/03/PrologAndWatson1.pdf
[20] http://www.cs.nmsu.edu/ALP/2011/03/natural-language-processing-with-prolog-in-the-ibm-watson-system/

in functional-style "MapReduce" schemes of computations allowing declarative high-performance concurrent processing over massive amounts of data,[21] that the $\mathcal{AWOL}$ project intends to adapt and use.

Constraints offer a natural declarative paradigm for concurrent distributed computing when a process is construed as constraint normalization [43]. The constrained logical variables shared by several constraints lend themselves to automatic inter-process communication channels coordinating the concurrent normalization of all constraints. Normalization (the process of solving a constraint) transforms a piece of data structure wherein constrained logical variables occur until it reaches either (1) a *solved form*—in which case the constraint is verified (*i.e.*, it "succeeds"); or (2) an *inconsistent* form—in which case the constraint is violated (*i.e.*, it "fails"); or (3) an *undecided* form—in which case the constraint normalization process is suspended until additional information accrues (*i.e.*, it "residuates").[22] Thus, computation proceeds as the simultaneous normalization of constraints. In this (implicit) way, processes sharing logical variables use them as communication channels [6].

A well-known example of contraint solving is First-Order Term (FOT) *matching* as used in FP (*e.g.*, term-rewriting systems). It consists in the normalization of the constraint that "$t_2$ must be more general than (or *subsume*) $t_1$." A FOT matching constraint solved form is a finite conjunction of variable bindings of $t_2$ (a subsitution $\sigma$ such that $t_1 = t_2\sigma$).

Another well-known example is FOT *unification* as used in LP (*e.g.*, Prolog). Indeed, FOT unification of two FOTs $t_1$ and $t_2$ consists in the normalization of the equality constraint $t_1 = t_2$. A unification's solved form is a finite conjunction of variable bindings of both FOTs' logical variables (a subsititution $\sigma$ such that $t_1\sigma = t_2\sigma$).

A lesser-known example is FOT unification modulo functional dependencies (as used in LeFun [17, 16]). There, since function application uses FOT matching while predicate resolution uses FOT unification, function application may suspend and wait until arguments become sufficiently instantiated (through concommitant unifications).

Yet another constraint system generalizing FOT matching and unification is $\mathcal{LIFE}$'s $\mathcal{OSF}$ graph matching and unification [18, 21]. It is this latter constraint system that we propose to generalize further in the $\mathcal{AWOL}$ project to a fully distributed concurrent system to provide efficient ontological reasoning power for the Semantic Web.

More expressive $\mathcal{OSF}$-based systems can be used for ontological reasoning, in which concepts in taxonomies are formalized as set-denoting sorts subject to arbitrary relational and functional constraints. Depending on the expressiveness, so-constrained $\mathcal{OSF}$ unification can become undecidable. However, decidable restrictions thereof still offer very expressive capabilities for ontological reasoning [20]. Even more interesting is the use of *lazy* constraint normalization [3]. This technique guarantees that only constraints relevant to a concept's use be normalized. This adds efficiency as well as tolerating potentially undecidable constraints, although at the expense of losing completeness. In addition, implementation techniques generalizing the way Logic Programming is compiled into low-level abstract machine instructions (*viz.*, [1]) are readily adaptable [12, 22].

### What does $\mathcal{AWOL}$ have to offer that hasn't been offered yet?

The $\mathcal{AWOL}$ project's proposed approach is a radically different alternative to the majority of current Semmantic Web systems which almost exclusively rely on Description Logic. Because the latter paradigm uses Tableau-based reasoning, it is at best very difficult to scale up reasoning and data analysis to the sizes of knowledge and data bases that accrue, and will continue to do so "*explonentially*" [10]. It addition, incremental and fault-tolerant distributed reasoning is still an unsolved set of challenges for the OWL-family of ontology web languages.

Even at the syntax level, OWL is, by any standards, unwieldy and confusing. However, having a simple uniform syntax using a universal information structure is an important aspect of any AI language. From LISP's S-expression to Prolog's First-Order Term (FOT), it enables meta-programming through a quote/eval mechanism. It is thus easy to define dynamic syntax-generating macros, a great tool most appreciated by seasoned AI programmers.[23] In $\mathcal{LIFE}$, this facility was put to use everywhere in the generation of both static and dynamic code. For Natural Language Processing (NLP) applications, for example, Prolog's Definite-Clause Grammars (DCG)

[21] Technically: monoid homomorphisms [27].

[22] The term "*residuation*" and verb "*to residuate*" were coined by the PI to denote a residual yet unsolved constraint (since it is a computational residue) [17, 16]. Residual variables are those yet unbound remaining in one or several residuations.

[23] http://docs.julialang.org/en/release-0.1/manual/metaprogramming/

are but a meta-syntax preprocessing into Prolog's syntax with continuation structures.[24] Even more expressive multidimensional DCG-like preprocessing is possible (as shown in $\mathcal{LIFE}$) thanks to $\mathcal{OSF}$ structures using so-called *accumulators*. This is done thanks to paired in/out feature labels that can discriminate among distinct information-processing threads, compose them, and invert them.[25]

But a new lower-level representation in the form of graphs (sets of RDF triples) is now the standard representation. Despite insignificant quibbles on notation (sets of triples *vs.* sets of key/value pairs), the consensus *de facto* is undoubtedly that a *graph data model* now prevails as the vehicle of information.[26] And such is the case at all levels: whether type/schema/TBox or program/data/Abox.

Finally, and quite importantly, the RDF has been adopted as a new data format replacing the relational model. Just as SQL became the technology for relational databases, so has the SPARQL RDF Query Language become the standard RDF-querying and view-generating language. Combining the reasoning power of $\mathcal{HOOT}$ on TBoxes to optimize the use of SPARQL on Aboxes as the RDF-oriented processing of massive triplestores can also be used to pragmatic advantage with $\mathcal{HOOT}$ [11].

### Related work

A noteworthy offshoot of several key ideas originating in the $\mathcal{LIFE}$ project, is the work done by Gert Smolka on the Oz language [46], and later re-designed and re-implemented by Peter Van Roy as the Mozart programming system.[27] Both Smolka and Van Roy had been my former collaborators in the $\mathcal{LIFE}$ project at Digital Paris Research Lab from 1988 until 1994.

More recently, and sharing basic ideas with Mozart's and our approach to concurrent constraint processing using logical variables and process communication channels, but exploited in the area of service-oriented programming, is the SyncFree project coordinated by Marc Shapiro.[28] SyncFree's basic idea, the notion of a *Commutative Replicated Data Type* (CRDT), is very related to incremental distributed constraint programming (through the use of "monotonic distributed data structures") [38]. In fact, the distributed object unification operation of Distributed Oz is a (simple) CRDT. Each replicated node does local unification, and as long as there are no unification failures, all nodes are consistent without communicating with each other.

Yet another offshoot of the Mozart system is the Transreal Initiative.[29] This was submitted as a Future and Emerging Technologies (FET) proposal that did not get funded.[30] It has the interesting notion of *elasticity*—the ability to ramp up resources quickly to meet inciental demand (like, *e.g.*, an electric plant).

Both CRDTs (commutative data types) and elasticity (adaptive clouds) are in perfect agreement with our approach to distributed $\mathcal{OSF}$-based constraint programming in that they seek to process information incrementally as it materialize.

### Implementation consideration

**Scala/Akka** To its designers and many other language experts and users, and this author as well, Scala is best described as "*Java/C# done right*."[31] Some striking advantages of Scala is that is has powerful features unsupported by Java (nor C#) while staying fully cross-compatible with Java through the use of a JVM backend. Among the most useful to ease implementation of several $\mathcal{AWOL}$–$\mathcal{HOOT}$ features, Scala provides:

- a unified type system (value and object types all substype of `Any`) makes worrying about value/object boxing/unboxing unecessary;
- primitive comprehensions (`for`-expressions);

[24] Such a linear-list representation, then called *difference-list*, is just a pair of logical variables `X-Y`: one (`X`) bound to its start and one (`Y`) unbound used as sequence terminator rather than using the empty list `[]` as usual. Thus, any difference-list `X-X` with the same stat and end variable is the empty sequence. This representation makes sequence concatenation constraints solvable in constant time by simply binding the end-variable of the first to the start of the second.

[25] http://hassan-ait-kaci.net/pdf/WildLIFE-HANDBOOK.pdf

[26] See for example the standardized triple data formats and inter-format mappers: http://rdf-translator.appspot.com/.

[27] http://mozart.github.io/mozart-v1/doc-1.4.0/dstutorial/index.html

[28] This is a 3-year European project that started October 2013.

[29] http://beernet.info.ucl.ac.be/staticfiles/transreal/IntelligentSystemsFuture.pdf

[30] It passed threshold but not high enough.

[31] The name "Scala" stands for "Scalable language."

- higher-order functional types and methods are supported;
- partial application of functional (currying) returning higher-order functions is automatic (nameless functions—*i.e.*, closures—are supported);
- lazy (delayed) expressions;
- delimited continuations (non-local exits); and,
- a powerful concurrent processing library (based on the Actor Model).

This last point, the most recent, is a powerful addition to Scala's set of libraries: the Akka actor-model concurrency toolset. The Actor Model[32] is a very flexible and powerful concurrent programming model.

> Its development was "motivated by the prospect of highly parallel computing machines consisting of dozens, hundreds or even thousands of independent microprocessors, each with its own local memory and communications processor, communicating via a high-performance communications network" (Carl Hewitt). Since that time, the advent of massive concurrency through multi-core computer architectures has revived interest in the Actor model.

The Actor Model is the concurrent paradigm used in the Erlang programming language,[33] and, as stressed, of the more recent Scala.[34,35]

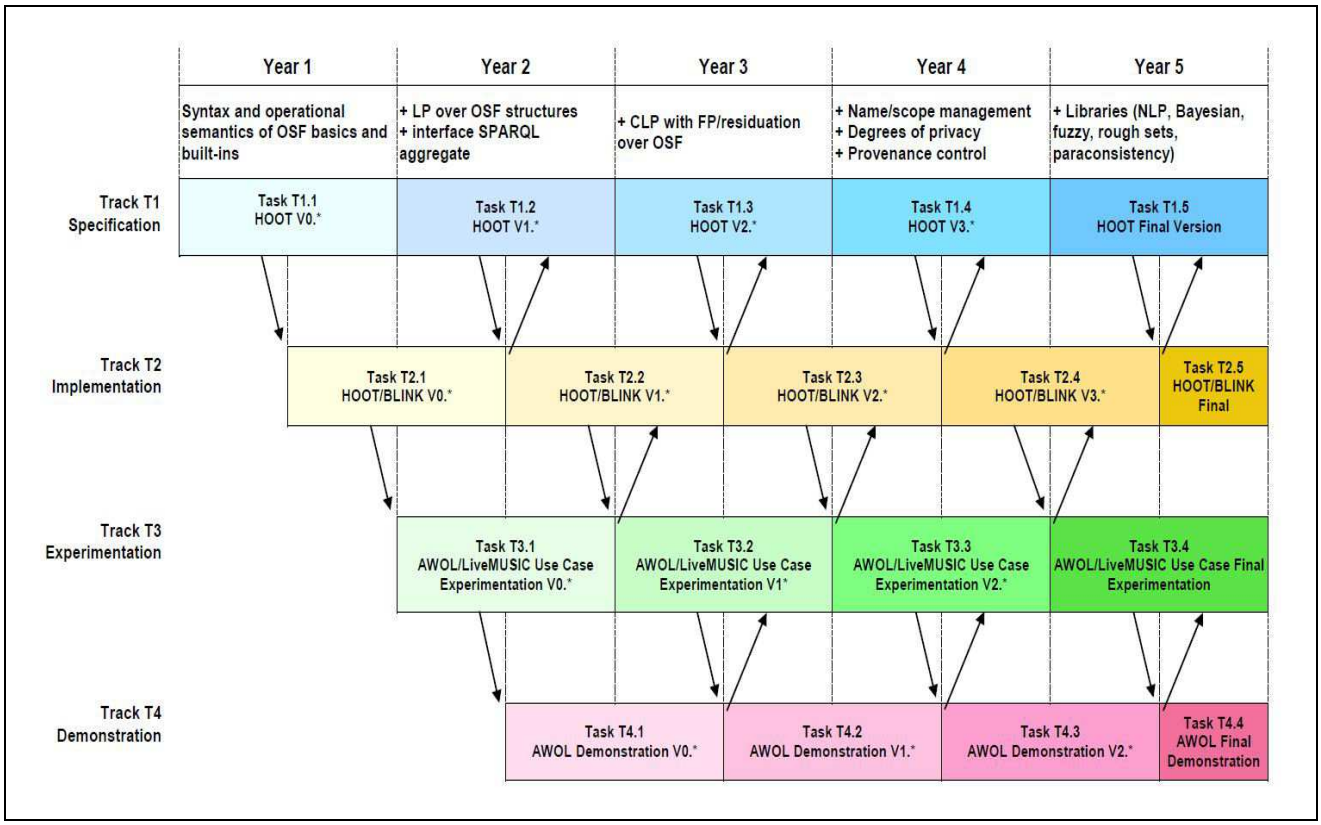
**X10** An alternative to Scala, also open-source, could be IBM Research's X10 multicore language.[36] In fact, X10 is the implementation language of C10, a strongly-typed, object-oriented, probabilistic, timed, Concurrent Constraint Programming language currently being designed and developed by Vijay Saraswat of IBM Research.[37]

X10 is surprisingly close to Scala: it is a polymorphically typed higher-order object-oriented language functional with a Scala-like syntax. It compiles to the JVM (through Java) and C++. It natively supports concurrent constraint programming. Its concurrency model, the Asynchronous Partitioned Global Address Space (APGAS) Programming Model,[38] is based on the notion of *places* and *asynchrony*.[39] An X10 statement qualified with the keyword "`async`" is spawned as new process, a so-called "*activity*," running in parallel and returns immediately. Such activities communicate through the variables they share. A synchronizing construct "`finish`" preceding a statement forces waiting for all subactivities to end before proceding. In a way, "`async`" is similar, though not quite, to the "`lazy`" expression constructs in Scala. However there is no need in Scala for explicit synchronization: it is automatic. An X10 activity corresponds essentially to an actor in the Actor Model. The notion of *place*—a collection of data and activities over such data—is a means to do concurrent computation on a distributed shared memory. The core API provides a global structure (named "`GlobalRef`") that can be used to migrate computation from a current local place to another local place "`p`" within the global space with the construct "`at (p)`" preceding a statement. This suspends its execution in its current place, serializes local variable and the data they are bound to, and move them to place "`p`," where they are deserialized and execution resumes.

Compared to X10, Scala offers simpler and more powerful constructs. Along with the Akka library, it comes to be a better choice of implementation vehicle for a more convenient management of concurrency with distributed shared variables. On the other hand, X10's native support of distributed concurrent constraint-processing sharing a partitioned global address space allows it to run on specialized parallel hardware (*e.g.*, GPGPU) to boost performance.

The foregoing facts justify the PI's preference for using Scala (possibly along with Java) as the implementation language for the $\mathcal{AWOL}$ project. It is a well-designed language, and makes a clever opportunistic use of the JVM as a machine language. Scala's Akka library provides actor-based multi-threading management of concurrent processes. This, with Scala itself and its other libraries, make it a convenient toolkit for the specific needs of the $\mathcal{AWOL}$ project's constraint-based reasoning system.

[32] https://en.wikipedia.org/wiki/Actor_model
[33] http://www.erlang.org/
[34] http://www.scala-lang.org/
[35] http://en.wikipedia.org/wiki/Scala_(programming_language)
[36] http://x10-lang.org/
[37] https://github.com/saraswat/C10
[38] http://x10-lang.org/documentation/tutorials/apgas-programming-in-x10-24.html
[39] http://x10.sourceforge.net/documentation/intro/latest/html/node4.html




**$\mathcal{AWOL}$ Project's Gantt-Chart Task Schedule**

# Scientific Program

## Project organization

**Task descriptions** The core of the $\mathcal{AWOL}$ project will consist in four concurrent tracks:

1. **Track T1—Specification** Tasks of this track will focus on activities relating to providing the linguistic details of the various operational parts of the $\mathcal{AWOL}$ system. It consists of a language with a syntax and operational semantics for specifiying, verifying, and reasoning with ontological taxonomies. It will be an elaboration of the $\mathcal{HOOT}$ language [7], the basic ontological language proposed in the course of the $\mathcal{CEDAR}$ project. These parts make up a language with a concrete syntax and an operational semantics consisting of knowledge-processing units (*i.e.*, related to the syntax and meaning of the contents of an Assertional Box—or ABox); of data-processing units (*i.e.*, related to the syntax and meaning of the contents of a Terminological Box—or TBox); and of query-processing over specific knowledge and data (*i.e.*, related to the syntax and meaning of the contents of a Query Box—or QBox). They will be packaged as a sequence of gradually more sophisticated system designs of data/knowledge processing for ontological programming. The targeted execution context for the $\mathcal{HOOT}$ language will be the $\mathcal{BLINK}$ distributed architecture [8] (or similar distributed file system), possibly with simulations of eventual missing functionalities.
2. **Track T2—Implementation** Tasks in this track will make up the sequence of partial implementations of the gradually more complex designs from Track T1. It is expected to provide also feedback to Track T1 for potential language design adjustments made apparent from concrete implementation experience. It is important to realize that the intent of the $\mathcal{AWOL}$ project is to target specifically the $\mathcal{BLINK}$ distributed architecture, and experiment with its prototype as a system-level intermediate layer for use cases involving massive amounts of data [8]. It may simulate such an interface as the implementation context may be. However, the use of $\mathcal{BLINK}$ *per se* is not mandatory: any similar distributed file system can be used. Another important point is the candidate's deliberate choice of using Scala as the prototyping language vehicle for the justifications spelled out in the preceding section.

3. **Track T3—Experimentation** Tasks in this track will make up the sequence of testing partial implementations realized in Track T2 as they materialize over experimental knowledge and data bases. This track will too provide feedback to the precedent (Track T2) for potential adjustments made apparent from actual benchmarks and concurrent orchestration protocols (including network-oriented communication). Experimentation on benchmarks—such as the the Lehigh University Benchmark (LUBM), or the Berlin SPARQL Benchmark (BSBM), but also on our own large-attributed TBox generator—generating queries on domain/range and other (relational and functional) constraints on attribute for huge existing ontologies (*e.g.*, NCBI). The development of a frontend OWL-to-$\mathcal{HOOT}$ compiler to enable processing OWL-based ontologies in $\mathcal{HOOT}$.
4. **Track T4—Demonstration** Tasks in this track will demonstrate the $\mathcal{AWOL}$ system's behavior as deployed on actual knowledge/data networked over the Internet. Several use cases will be considered and measurement made in comparison with the (then available) state of the art.

More details on each track and the tasks comprising them follow.

**T1 Specification Track—total duration: 20 quarters** (4 quarters = 1 year)

- **Task T1.1**—$\mathcal{HOOT}$ V0.* syntax and operational semantics **Duration**: 4 quarters.
  - basic sort taxonomies; Modules; modulated subsort encoding; incremental encoding;
  - syntax-driven operational semantics using Plotkin Structural Operational Semantics [41, 42];
  - feature domain/range constraints; down-propagation of constraints in the taxonomy;
  - implementation pragmatics; pre-processing; normalization; compilation; efficiency considerations; built-in aspects; shared-scope communication; communication among solvers;
  - develop specifications for typical use cases of $\mathcal{HOOT}$ V0.* illustrating the extents (and limits) of its expected expressive and operational power.

- **Task T1.2**—$\mathcal{HOOT}$ V1.* syntax and operational semantics **Duration**: 4 quarters. Additional features of $\mathcal{HOOT}$ V1 *vs.* $\mathcal{HOOT}$ V0:
  - add net-aware relational constraints over $\mathcal{OSF}$ graph structures and LOGIN-like rules,[40] and CLP semantics to define semantic properties of sorts defined in a taxonomy;
  - down-propagation of rule-defined relational constraints in the taxonomy, and ontology normalization (preprocessing and compilation);
  - add SPARQL interface with knowledge-driven query optimization;[41]
  - develop specifications for typical use cases of $\mathcal{HOOT}$ V1.* illustrating the extents (and limits) of its expressive and operational power.

- **Task T1.3**—$\mathcal{HOOT}$ V2.* syntax and operational semantics **Duration**: 4 quarters. Additional features of $\mathcal{HOOT}$ V2 *vs.* $\mathcal{HOOT}$ V1:
  - Add functional-dependency constraints over $\mathcal{OSF}$ graph structures, using net-aware *residuation* [17, 16] as an implicit communication mechanism;
  - down-propagation of rule-defined functional constraints in the taxonomy;
  - develop specifications for typical use cases of $\mathcal{HOOT}$ V2.* illustrating the extents (and limits) of its expressive and operational power.

- **Task T1.4**—$\mathcal{HOOT}$ V3.* syntax and operational semantics **Duration**: 4 quarters. Additional features of $\mathcal{HOOT}$ V3 *vs.* $\mathcal{HOOT}$ V2:
  - management of net-aware name scopes; degrees of privacy;
  - management of uncertainty (Bayesian inference, fuzzy-set logic, rough-set, paraconsistency, . . . )

[40] Net-aware Horn clauses over $\mathcal{OSF}$ terms [14, 15].
[41] Extending the technique used in [11].

  - develop specifications for typical use cases of $\mathcal{HOOT}$ V2.* illustrating the extents (and limits) of its expressive and operational power.

- **Task T1.5**—final $\mathcal{HOOT}$ syntax and operational semantics **Duration**: 4 quarters. Ultimate version to meet the $\mathcal{AWOL}$ project's language-level ($\mathcal{HOOT}$ V4.*). Additional features of $\mathcal{HOOT}$ V4 *vs.* $\mathcal{HOOT}$ V3:

  - ensure stable version syntax and semantics;
  - full recap specification.

**T2 Implementation Track—total duration: 18 quarters** Implementation Track T2 parallels Specification Track T1 with 6 months delay. Each task corresponds to implementing the specifications of the corresponding task in Track T1, as illustrated on the task schedule diagram of Figure .

- **Task T2.1**—$\mathcal{HOOT}$ on $\mathcal{BLINK}$ V0.* implementation of $\mathcal{HOOT}$ V0.* specified up to the end of Task T1.1. **Duration**: 4 quarters
- **Task T2.2**—$\mathcal{HOOT}$ on $\mathcal{BLINK}$ V1.* implementation of $\mathcal{HOOT}$ V1.* specified up to the end of Task T1.2. **Duration**: 4 quarters
- **Task T2.3**—$\mathcal{HOOT}$ on $\mathcal{BLINK}$ V2.* implementation of $\mathcal{HOOT}$ V2.* specified up to the end of Task T1.3. **Duration**: 4 quarters
- **Task T2.4**—$\mathcal{HOOT}$ on $\mathcal{BLINK}$ V3.* implementation of $\mathcal{HOOT}$ V3.* specified up to the end of Task T1.4. **Duration**: 4 quarters
- **Task T2.5**—final $\mathcal{HOOT}$ on $\mathcal{BLINK}$ implementation of final $\mathcal{HOOT}$ version specified up to the end of Task T1.5. **Duration**: 2 quarters

**T3 Experimentation Track—total duration: 16 quarters** Experimentation Track T3 parallels Implementation Track T2 with 6 months delay. Each task corresponds to experimentng with the implementation developed in the corresponding task in Track T2, as illustrated on the task schedule diagram of Figure .

- **Task T3.1**—$\mathcal{LIVEMUSIC}$ use case in $\mathcal{HOOT}$ V0.* experimentation with $\mathcal{HOOT}$ on $\mathcal{BLINK}$ implemented up to the end of Task T2.1. **Duration**: 4 quarters
- **Task T3.2**—$\mathcal{LIVEMUSIC}$ use case in $\mathcal{HOOT}$ V1.* experimentation with $\mathcal{HOOT}$ on $\mathcal{BLINK}$ implemented up to the end of Task T2.2. **Duration**: 4 quarters
- **Task T3.3**—$\mathcal{LIVEMUSIC}$ use case in $\mathcal{HOOT}$ V2.* experimentation with $\mathcal{HOOT}$ on $\mathcal{BLINK}$ implemented up to the end of Task T2.3. **Duration**: 4 quarters
- **Task T3.4**—final $\mathcal{LIVEMUSIC}$ use case in $\mathcal{HOOT}$ experimentation with final $\mathcal{HOOT}$ on $\mathcal{BLINK}$ implemented up to the end of Task T2.4. **Duration**: 4 quarters

**T4 Demonstration Track—total duration: 14 quarters** Demonstration Track T4 parallels Experimentation Track T3 with 6 months delay. Each task corresponds to demonstrating with the results honed in the experimentation performed in the corresponding task in Track T3, as illustrated on the task schedule diagram of Figure .

- **Task T4.1**—$\mathcal{AWOL}$ System V0.* demonstration of $\mathcal{LIVEMUSIC}$ use case in $\mathcal{HOOT}$ V0.* experimented with up to the end of Task T3.1. **Duration**: 4 quarters
- **Task T4.2**—$\mathcal{AWOL}$ V1.* demonstration of $\mathcal{LIVEMUSIC}$ use case in $\mathcal{HOOT}$ V1.* experimented with up to the end of Task T3.2. **Duration**: 4 quarters
- **Task T4.3**—$\mathcal{AWOL}$ System V2.* demonstration of $\mathcal{LIVEMUSIC}$ use case in $\mathcal{HOOT}$ V2.* experimented with up to the end of Task T3.3. **Duration**: 4 quarters
- **Task T4.4**—final $\mathcal{AWOL}$ System demonstration of $\mathcal{LIVEMUSIC}$ use case in $\mathcal{HOOT}$ V3.* experimented with up to the end of Task T3.4. **Duration**: 2 quarters

**Description of requested resources**

**Personnel other than PI**

**Post-docs** Two high-caliber internationally competitive post-doc positions: one overseeing Tracks 1 and 2 and one overseeing Tracks 3 and 4.

- **Specification and implementation** Prerequisites: alumnus of top-rated institutions; solid experience in language design and prototyping; solid practical experience in software system development in Java, C++, C (Scala a plus), as well as scripting (JavaScript, Python, C shells, *etc.*); working familiarity with the theory and practice of Constraint Logic Programming, Functional Programming, and Object-Oriented Programming; experience with Semantic Web technology; industrial software development a plus. Duties: overseeing Tasks T1 and T2. (**Duration**: 20 quarters)
- **Experimentation and demonstration** Prerequisites: alumnus of top-rated institutions; solid practical experience in software system development in Java, C++, C (Scala a plus), as well as scripting (JavaScript, Python, C shells, *etc.*); high familiarity with distributed database and RDF-based Web Programming tools, (esp. SPARQL); deep understanding of distributed database technology and tools; high familiarity with Big Data technology and tools; industrial software development a plus. Duties: overseeing Tasks T3 and T4. (**Duration**: 16 quarters)

**Research engineer** One highly-qualified Research Engineer in advanced software development.[42] Prerequisites: alumnus of top-rated institutions; solid practical experience in software system development in Java, C++, C (Scala a plus), as well as scripting (JavaScript, Python, C shells, *etc.*); serious experience in industrial software research protoping. Duties: in charge of setting up and maintaining the project's web site; $\mathcal{AWOL}$ software tools (*e.g.*, GUI, cloud access, *etc.*); software modules' cohesiveness; adherence to standards; general software (all 4 tracks). (**Duration**: 20 quarters)

**PhDs** Four PhD's positions: one per track.

- **PhD Topic 1** Formal design and sepcification of the $\mathcal{HOOT}$ knowledge representation and automated reasoning system, including an OWL-to-$\mathcal{HOOT}$ front-end compiler—Track T1. (**Duration** 12 quarters)
- **PhD Topic 2** Formal design and implementation of a Scala-based concurrent architecture for $\mathcal{HOOT}$, with a Scala/SPARQL interface and approximated reasoning libraries—Track T2. (**Duration** 12 quarters)
- **PhD Topic 3** Experimental design and investigation of a backend compilation scheme and concurrency protocols from the $\mathcal{HOOT}$'s Scala backend to a distributed file system ($\mathcal{BLINK}$, or other)—Track T3. (**Duration** 12 quarters)
- **PhD Topic 4** Design and development of a comprehensive $\mathcal{HOOT}$ demonstration platform on actual benchmarks—Track T4 (**Duration** 12 quarters)

**MSc interns** Four MSc positions: one per track.

- **MSc Topic 1** Realization and testing of a high-performance Scala library for partial-order encoding, including, but not only, bit-vector encoding; code modulation; incremental encoding—Track T1. (**Duration** 4 quarters)
- **MSc Topic 2** Realization and testing of a knowledge-enhanced $\mathcal{HOOT}$/SPARQL interface for high-performance intelligent Big Linked Data processing—Track T2. (**Duration** 4 quarters)
- **MSc Topic 3** Experimental investigation of a backend compilation scheme and concurrency protocols for $\mathcal{HOOT}$ to a distributed file system (from Scala to $\mathcal{BLINK}$, or similar distributed file system)—Track T3. (**Duration** 4 quarters)
- **MSc Topic 4** Contribution to the $\mathcal{HOOT}$ demonstration platform on actual cloud benchmarks—Track T4. (**Duration** 4 quarters)

**Other necessary resources**

- **Travel**—estimated national/international travel (conferences, meetings, scientific guests), for 7 persons (project members or visitors) per year.
- **Other consumables**—extra memory per laptop/desktop for all project members.
- **Other goods and services**—organization of 3 scientific meetings over the course of 5 years.

[42] In French: "*ingénieur de recherche*" ideally member of "*Corps des Mines*" or similar excellence.

- **Subcontracting**—access to cloud network platforms for experimentation with distributed ontological processing over distributed triplestores.

### Potential collaborations

The $\mathcal{AWOL}$ system will benefit from potential research collaborations with prominent international figures. No remuneration or other compensation is envisaged for such collaborations. The persons listed below are all esteemed peers, senior figures in their field. Each has agreed on the principle to contribute a small amount of their time to collaborate with the $\mathcal{AWOL}$ project in the form of student co-supervision or internship on specific topics as appropriate.

- **Pat Lincoln, SRI International**—I have known Pat Lincoln, now Director of SRI International's CS Lab, since he was part of the research project that I led at MCC, in Austin, TX, in the late 1980's. He has since made quite a name for himself in a varied array of advanced Computer Science research areas—*viz.*, AI, computational biology, and cyber-security. He was one of the original implementers of the first ever version of $\mathcal{LIFE}$ at MCC.
- **Manuel Hermenegildo, Universidad Politécnica de Madrid**—A former colleague of mine at MCC, in Austin, TX, Manuel Hermenegildo, is now a Professor at the Universidad Politécnica de Madrid (UPM), as well as founder and director of the IMDEA Software Institute, also in Madrid. He is considered among the foremost *aficionados* in parallel constraint logic programming. He is the principal designer and mastermind of the CIAO CLP programming system at UPM's CLIP Lab, a research group he created in, and has headed since, the late 1990's. Among many of Manuel's original ideas, CIAO also integrates some ideas from $\mathcal{LIFE}$.
- **Peter Van Roy, Université Catholique de Louvain**—Peter Van Roy is a former member of the $\mathcal{LIFE}$ development team at Digital PRL. He is now a Professor at the Université Catholique de Louvain (UCL). He has been a major contributor to the theory and practice of Concurrent Constraint Programming and is now a prominent figure in the field. His work on the Mozart constraint programming system is highly regarded, and borrowed several key ideas from our previous common work on $\mathcal{LIFE}$. Lately, his interests have grown to address similar concerns as some expressed in this proposal.
- **Gopalan Nadathur, University of Minnesota**—Gopalan Nadathur is a professor in the department of Computer Science and Engineering at the University of Minnesota. He is one of the early designers of the higher-order logic programming language $\lambda$Prolog, and the main designer of its Teyjus implementation. He is quite familiar with my work, and $\mathcal{LIFE}$ in particular. We have known each other since we were both doing our PhDs at the University of Pennsylvania.

## Conclusion

I have presented the technical context and methodology I am proposing for the $\mathcal{AWOL}$ project. I believe that what I am proposing is a bold and ambitious challenge—and, as well, a maverick's pursuit. Because of its defiant nature,[43] this proposal is a radical departure from mainstream research and development of a truly *semantic* web. In this regard, it is admittedly a "high-risk" proposal. However, it is also definitely a "high-gain" endeavor as well if successful. Confidence that it is feasible comes, first of all, from my own experience having designed and overseen the many years of internationally recognized high-quality research and development efforts that led to the $\mathcal{LIFE}$ constraint-logic programming language. I am also and especially comforted by the encouraging experimental results obtained in the course of the ANR Chair of Excellence $\mathcal{CEDAR}$ project in distributed ontological reasoning. Finally, by enabling constraint-driven reasoning such as described in this proposal, I trust my capacity to show the world that expressing knowledge with $\mathcal{HOOT}$ on a $\mathcal{BLINK}$ distributed system can process net-aware ontologies much more efficiently and scalably than any OWL may hope to. The expected result is a highly performant and scalable $\mathcal{A}$lternative $\mathcal{W}$eb $\mathcal{O}$ntology $\mathcal{L}$anguage.

[43] Essentially, my saying that the vast majority of current Semantic Web reasoners following the same non-scalable and unduly complicated Description-Logic gospel is a serious technological mistake.

# References


[1] AÏT-KACI, H. *Warren's Abstract Machine: A Tutorial Reconstruction*. The MIT Press, Cambridge, MA, USA, 1991. (online).

[2] AÏT-KACI, H. An introduction to LIFE—Programming with Logic, Inheritance, Functions, and Equations. In *Proceedings of the International Symposium on Logic Programming* (October 1993), D. Miller, Ed., MIT Press, pp. 52–68. (online).

[3] AÏT-KACI, H. Data models as constraint systems—a key to the Semantic Web. *Constraint Processing Letters 1* (November 2007), 33–88. (online).

[4] AÏT-KACI, H. Children's magic won't deliver the Semantic Web. Letter to the Editor, Communications of the ACM, March 2009. (online).

[5] AÏT-KACI, H. Is it possible to make the Semantic Web a reality? Keynote presentation at the 3rd edition of the international workshop on Innovation and New Trends in Information Systems (INTIS 2013), Tangiers, Morocco, November 22–23, 2013. (online).

[6] AÏT-KACI, H. Reasoning and the Semantic Web. Invited presentation at the Ontolog Forum's Panel on Ontology, Rules, and Logic Programming for Reasoning and Applications (RulesReasoningLP) mini-series of virtual panel sessions, November 21, 2013. (online).

[7] AÏT-KACI, H. $\mathcal{HOOT}$: A language for expressing and querying hierarchical ontologies, objects, and types—a specification. CEDAR Technical Report (forthcoming), $\mathcal{CEDAR}$ Project, LIRIS, *Département d'Informatique, Université Claude Bernard Lyon 1*, Villeurbanne, France, October 2014.

[8] AÏT-KACI, H. $\mathcal{BLINK}$—$\mathcal{B}$ig $\mathcal{L}$inked $\mathcal{I}$nformation as $\mathcal{N}$etworked $\mathcal{K}$nowledge. Proposal submitted to the ANR @TRaction 2014 Program, June 2014.

[9] AÏT-KACI, H. Conceptual roles as set-valued order-sorted features—relations as functional aggregates. CEDAR Technical Report (forthcoming), $\mathcal{CEDAR}$ Project, LIRIS, *Département d'Informatique, Université Claude Bernard Lyon 1*, Villeurbanne, France, October 2014.

[10] AÏT-KACI, H., AND AMIR, S. Classifying and querying very large taxonomies—a comparative study to the best of our knowledge. CEDAR Technical Report Number 2, $\mathcal{CEDAR}$ Project, LIRIS, *Département d'Informatique, Université Claude Bernard Lyon 1*, Villeurbanne, France, May 2013. (online).

[11] AÏT-KACI, H., AND AMIR, S. Design and implementation of an efficient semantic web reasoner. CEDAR Technical Report Number 12, $\mathcal{CEDAR}$ Project, LIRIS, *Département d'Informatique, Université Claude Bernard Lyon 1*, Villeurbanne, France, October 2014. (online).

[12] AÏT-KACI, H., AND DI COSMO, R. Compiling order-sorted feature term unification. PRL Technical Note 7, Digital Paris Research Lab, Rueil-Malmaison, France, December 1993. (online).

[13] AÏT-KACI, H., HACID, M.-S., HAQUE, R., AND FOURURE, D. Experiments with scalable triplestores. CEDAR Technical Report Number 5, $\mathcal{CEDAR}$ Project, LIRIS, *Département d'Informatique, Université Claude Bernard Lyon 1*, Villeurbanne, France, October 2013. (online).

[14] AÏT-KACI, H., AND NASR, R. Logic and inheritance. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL'86)* (Saint-Petersburg, FL, USA, January 1986), pp. 219–228. (online).

[15] AÏT-KACI, H., AND NASR, R. LOGIN: A logic programming language with built-in inheritance. *Journal of Logic Programming 3* (1986), 185–215. (online).

[16] AÏT-KACI, H., AND NASR, R. Integrating logic and functional programming. *Lisp and Symbolic Computation 2* (1989), 51–89. (online).

[17] AÏT-KACI, H., NASR, R., AND LINCOLN, P. Le Fun: Logic, equations, and Functions. In *Proceedings of the Symposium on Logic Programming (San Francisco, CA)* (Washington, DC, 1987), IEEE, Computer Society Press, pp. 17–23. (online).

[18] AÏT-KACI, H., AND PODELSKI, A. Towards a meaning of LIFE. *Journal of Logic Programming 16*, 3-4 (1993), 195–234. (online).

[19] AÏT-KACI, H., AND PODELSKI, A. Functions as passive constraints in LIFE. *ACM Transactions on Programming Languages and Systems 16*, 4 (July 1994), 1279–1318. (online).

[20] AÏT-KACI, H., AND PODELSKI, A. Order-sorted feature theory unfication. *Journal of Logic Programming 20*, 2 (1997), 99–124. (online).

[21] AÏT-KACI, H., PODELSKI, A., AND SMOLKA, G. A feature constraint system for logic programming with entailment. *Theoretical Computer Science 122* (1994), 263–283. (online).

[22] AÏT-KACI, H., AND VORBECK, M. Compiling order-sorted feature unification. Unpublished Technical Report, February 1996. (online).

[23] AMIR, S., AND AÏT-KACI, H. $\mathcal{CEDAR}$: a fast taxonomic reasoner based on lattice operationssystem demonstration. In *Proceedings of the Posters & Demonstrations Track of the 12th International Semantic Web Conference* (Sydney, Australia, October 2013), E. Blomqvist and T. Groza, Eds., CEUR Workshop Proceedings, pp. 9–12. (online).

[24] BAADER, F., BRANDT, S., AND LUTZ, C. Pushing the $\mathcal{EL}$ envelope. In *Proceedings of the 19th International Joint Conference on Artificial Intelligence* (Edinburgh, Scotland, UK, July-August 2005), L. P. Kaelbling and A. Saffiotti, Eds., IJCAI'05, Morgan Kaufmann Publishers, pp. 364–369. (online).

[25] BAADER, F., LUTZ, C., AND SUNTISRIVARAPORN, B. CEL—a polynomial-time reasoner for life science ontologies. In *Proceedings of the 3rd International Joint Conference on Automated Reasoning* (Seattle, WA, USA, August 2006), U. Furbach and N. Shankar, Eds., IJCAR'06, Springer-Verlag LNAI Vol. 4130, pp. 287–291. (online).

[26] CHEN, M., HAQUE, R., AND HACID, M.-S. CedTMart—a triplestore for storing and querying blinked data. CEDAR Technical Report Number 2, $\mathcal{CEDAR}$ Project, LIRIS, *Département d'Informatique, Université Claude Bernard Lyon 1*, Villeurbanne, France, July 2013. (online).

[27] FEGARAS, L., AND MAIER, D. Optimizing object queries using an effective calculus. *ACM Transactions on Database Systems 25* (1998), 25–4. (online).

[28] FIKES, R., HAYES, P., AND HORROCKS, I. OWL-QL—a language for deductive query answering on the Semantic Web. *Journal of Web Semantics 2*, 1 (December 2004), 19–29. (online).

[29] GOODWIN, J. Experiences of using OWL at the ordnance survey. In *Proceedings of the OWLED'05 Workshop on OWL: Experiences and Directions* (Galway, Ireland, November 11–12, 2005), B. C. Grau, I. Horrocks, B. Parsia, and P. Patel-Schneider, Eds., CEUR Workshop Proceedings Vol. 188. (online).

[30] HAARSLEV, V., HIDDE, K., MÖLLER, R., AND WESSEL, M. The RacerPro knowledge representation and reasoning system. *Semantic Web Journal 1* (March 2011), 1–11. (online).

[31] HAARSLEV, V., AND MÖLLER, R. RACER system description. In *Proceedings of the 1st International Joint Conference on Automated Reasoning* (Siena, Italy, June 2001), R. Gore, A. Leitsch, and T. Nipkow, Eds., IJCAR'01, Springer-Verlag, pp. 701–706. (online).

[32] HORROCKS, I., AND SATTLER, U. A tableau decision procedure for $\mathcal{SHOIQ}$. *Journal of Automated Reasoning 39*, 3 (July 2007), 249–276. (online).

[33] JAFFAR, J., AND LASSEZ, J.-L. Constraint logic programming. In *Proceedings of the 14th ACM Symposium on Principles of Programming Languages (POPL'87)* (New York, NY, USA, 1987), Association for Computing Machinery, pp. 111–119. (online).

[34] JAFFAR, J., AND MAHER, M. J. Constraint logic programming—a survey. *The Journal of Logic Programming 19* (July 1994), 503–581. (online).

[35] KAZAKOV, Y. Consequence-driven reasoning for horn $\mathcal{SHIQ}$ ontologies. In *Proceedings of the 21st International Conference on Artificial Intelligence* (Pasadena, CA, USA, July 2009), C. Boutilier, Ed., IJCAI'09, Association for the Advancement of Artificial Intelligence, pp. 2040–2045. (online).

[36] KAZAKOV, Y., KRÖTZSCH, M., AND SIMANČÍK, F. Unchain my $\mathcal{EL}$ reasoner. In *Proceedings of the 24th International Workshop on Description Logics* (Barcelona, Spain, July 2011), R. Rosati, S. Rudolph, and M. Zakharyaschev, Eds., DL'11, CEUR Workshop Proceedings. (online).

[37] LAWLEY, M. J., AND BOUSQUET, C. Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In *Proceedings of the 2nd Australasian Ontology Workshop: Advances in Ontologies (AOW 2010)* (Adelaide, Australia, December 2010), T. Meyer, M. A. Orgun, and K. Taylor, Eds., ACS, pp. 45–50. (online).

[38] LEŢIA, M., PREGUIÇA, N., AND SHAPIRO, M. CRDTs: Consistency without concurrency control. *Rapport de Recherche* N$^o$ 6956, INRIA, Rocquencourt, France, June 2009. (online).

[39] MANNA, Z., AND WALDINGER, R. Fundamentals of deductive program synthesis. In *Combinatorial Algorithms on Words*, A. Apostolico and Z. Galil, Eds., NATO ISI Series. Springer-Verlag, 1991. (online).

[40] MOTIK, B., SHEARER, R., AND HORROCKS, I. Hypertableau reasoning for description logics. *Journal of Artificial Intelligence Research 36*, 1 (September 2009), 165–228. (online).

[41] PLOTKIN, G. D. A structural approach to operational semantics. Tech. Rep. DAIMI FN-19, University of Århus, Århus, Denmark, 1981. (online).

[42] PLOTKIN, G. D. A structural approach to operational semantics. *Journal of Logic and Algebraic Programming 60–61* (January 30, 2004), 17–139. (online)—*N.B.*: Published version of [41].

[43] SARASWAT, V., AND RINARD, M. Concurrent constraint programming. In *Proceedings of the 17th Annual ACM Symposium on Principles of Programming Language (POPL'90)* (New York, NY, USA, 1990), F. E. Allen, Ed., Association for Computing Machinery, pp. 232–245. (online).

[44] SHEARER, R., MOTIK, B., AND HORROCKS, I. HermiT: A highly-efficient OWL reasoner. In *Proceedings of the 5th International Workshop on OWL Experiences and Directions* (Karlsruhe, Germany, October 2008), U. Sattler and C. Dolbear, Eds., OWLED'08, CEUR Workshop Proceedings. (online).

[45] SIRIN, E., PARSIA, B., GRAU, B. C., KALYANPUR, A., AND KATZ, Y. Pellet: A practical OWL-DL reasoner. *Journal of Web Semantics 5*, 2 (June 2007), 51–53. This is a summary; full paper: (online).

[46] SMOLKA, G. The Oz programming model. In *Computer Science Today*, J. van Leeuwen, Ed. Springer-Verlag, Berlin, Germany, 1995, ch. Lecture Notes in Compuer Science Vol. 1000, pp. 324–343. (online).

[47] SRINIVAS, K. OWL reasoning in the real world: Searching for Godot. In *Proceedings of the 22nd International Workshop on Description Logics (DL 2009)* (Oxford, United Kindgom, July 27–30 2009), B. C. Grau, I. Horrocks, B. Motik, and U. Sattler, Eds., CEUR Workshop Proceedings. Invited lecture (online).

[48] THOMAS, E., PAN, J. Z., AND REN, Y. TrOWL: Tractable OWL 2 reasoning infrastructure. In *Proceedings of the 7th Extended Semantic Web Conference* (Heraklion, Greece, May-June 2010), L. Aroyo, G. Antoniou, E. Hyvnen, A. ten Teije, H. Stuckenschmidt, L. Cabral, and T. Tudorache, Eds., ESWC'10, Springer-Verlag, pp. 431–435. (online).

[49] TSARKOV, D., AND HORROCKS, I. FaCT++ description logic reasoner: System description. In *Proceedings of the 3rd International Joint conference on Automated Reasoning* (Seattle, WA, USA, August 2006), U. Furbach and N. Shankar, Eds., IJCAR'06, Springer-Verlag, pp. 292–297. (online).