

# Statement of Research

Kenny Q. Zhu

March 1, 2009

I am interested in *programming technologies for data processing*. By data processing, I mean all tasks involving the accumulation, distribution, assembly, query, transformation and computation of data. In the past, I have studied the processing of two different kinds of data: *ad hoc data* and *knowledge bases*.

Ad hoc data refers to the billions of bytes of non-standard, semi-structured and continuously evolving data spread across all computer systems. Such data include server logs, performance and debugging information, telephone call records, financial data and repositories of scientific data. The formats of these data are widely varied and unpredictable, and are not standard like XML. Ad hoc data exists and will continue to exist far into the future for a variety of reasons: (1) it is compact (unlike XML, that can require 8 - 10 times as much space); (2) it is often easier to read for humans; (3) is produced by many legacy systems; and (4) programmers are “lazy” and usually do not craft well-structured schemas but simply output ad hoc data. Managing ad hoc data is a very difficult task because the data sources arrive “as is”, in large amounts, and may contain errors. Worse still, they are usually not well documented and do not have ready-made tools or libraries. In the past, dealing with ad hoc data is usually a matter of writing one-off parsers and tools in C or Perl for each data format, which is tedious, error-prone and hard to maintain.

A knowledge base is a collection of facts as well as rules to manipulate those facts. Data of this sort exists in domains like biology, medicine, pharmacy and manufacturing. It is usually not relational and not stored in relational databases. Knowledge base systems (KBS) are often implemented in logic programming languages such as Prolog and Datalog. Though there were a number of successful deployments of KBS in the 90's, KBS have yet to gain significant presence in mainstream applications. One of the critical limitations of traditional KBS is that they are *closed* and *not reactive*. In other words, these systems do not support an evolving set of agents or processes that interact with the knowledge base and each other.

As a postdoctoral researcher at Princeton, I have worked with David Walker (Princeton professor) and Kathleen Fisher (AT&T researcher) on the PADS project [7] and developed a format inference system called LEARNPADS [2, 1, 8] that automatically generates a suite of useful tools for processing ad hoc data, directly from sample data. More recently, with my co-authors, I have designed and implemented an extension to the PADS, named GLOVES[10], which is a system that generates monitoring, analysis and transformation tools for distributed ad hoc data from declarative specifications.

During my PhD studies at National University of Singapore (NUS), under the supervision of Joxan Jaffar and Roland Yap, I developed the Open Constraint Programming (OCP) system [4, 5, 6, 9], which implements a new programming paradigm for querying, updating and synchronizing with knowledge bases represented by constraint logic programs (CLP). Next, I will summarize my work on PADS and OCP and discuss some possible future directions.

## Processing of Ad Hoc Data

The goal of the PADS project (a joint project between Princeton and AT&T) is to improve the productivity of data analysts who need to cope with new and evolving ad hoc data sources on a daily basis. This is achieved through the design and development of languages and tools that help specify, query and manipulate ad hoc data. Our central technology is PADS, a domain-specific description language in which programmers can specify the structure and properties of data sources. The PADS compiler reads descriptions and produces a suite of programming libraries (parser, printer and validator) and end-to-end tools (XML translator, query engine, reformatter, error monitor, etc). However, the main impediment to using PADS is the time and expertise required to write a PADS description for a new data source. This can take anywhere from a few minutes to several days. Because ad hoc data sources often

come in massive volume but little or no documentation, understanding the structure of the data and producing a good description can be a daunting task.

To further improve productivity, along with my co-authors, I developed the LEARNPADS system [2, 1] in SML/NJ, which infers an accurate, human-readable PADS description from a small amount of sample ASCII data. The description can then be used to automatically generate tools and libraries for processing data of the same format. LEARNPADS incorporates a multi-phase inference algorithm whose main components are *tokenization*, *structure discovery* and *format refinement*. In tokenization, the sample data source is chunked into units of repetition such as lines or paragraphs, and then partitioned into simple tokens using a conventional lexer. The structure discovery phase computes the histograms of token frequencies, and predicts the structure of the data in a top-down, divide-and-conquer fashion. At the end of the phase, an initial rough structure is obtained in an intermediate representation form. Finally, in format refinement, a series of rewriting rules are applied to the initial structure to improve both the precision and the compactness of the description. The refinement is guided by an iterative hill-climbing procedure with a Minimum Description Length (MDL) score as its objective function. When the search reaches a local optimum, the description is produced in PADS and tools are generated. Our experiments on a variety of data sources showed that LEARNPADS was able to, on average, produce descriptions that are 95% accurate after training on just 5% of the original data. Moreover, the execution time scales linearly with the size of the sample data.

However, because we tokenize samples using tokens defined by fixed regular expressions, and parse a substring by the first matched token, the system was not able to disambiguate tokens that syntactically overlap with each other. For example, `integer` token and `float` token are ambiguous since string “88” be parsed as either `integer` or `float`. This is a problem we call *Token Ambiguity Problem (TAP)*. To solve TAP, we have recently redesigned the format inference algorithm [8] so that it takes advantage of information generated from an arbitrary statistical token model and select the tokens that are most probable to parse the data. In addition, we augment the refinement phase with a procedure that identifies complex, text-like structures in the inferred description and replace them with a simple `blob` token and thus simplify the final description. We evaluated the effectiveness of three different statistical models trained from a range of ad hoc data sources, and found the revised algorithm to produce much more accurate and compact descriptions.

So far I have discussed my work on processing ad hoc data from a single source such as a file. In practice, however, ad hoc data of different sorts are spread across many computer systems over the network, and more than often, system administrators, financial analysts and scientists need to gather these distributed data, integrate them and process them collectively. We developed the GLOVES system [10] that generates, from declarative specifications, tools and libraries for distributed ad hoc data processing. The generated tools include an archiver, a database loading system, a statistical analyzer, an alerter, an RSS feed generator, and debugging tools. The libraries include OCAML modules for printing, parsing, error management and data traversal, which developers can use to create their own custom tools. Advanced users can also built their own generic tools applicable to *any* collection of data sources. The GLOVES language allows the user to specify *where* the data is located, *how* to obtain it, *when* to get it and *what* preprocessing to do when it arrives. The language is layered on top of the PADS sub-language, which is used to specify the format of each individual data source. We used our new system to implement a prototype distributed system monitor inspired by the CoMon architecture. We have deployed it to monitor all the nodes on the PlanetLab testbed. Results showed that the GLOVES system is capable of supporting PlanetLab-scale monitoring applications.

## Open Constraint Programming

The OCP project was motivated by the challenge of providing *coordination* and *synchronization* to trading agents in automated marketplaces, which leads to more efficient and optimized transactions. A marketplace is governed by trading rules, and individual traders act based on their prior knowledge about the market which can also be formulated as logical rules. Therefore, a KBS is a natural choice for implementing such applications. However, traditional KBS's are written in sequential logic programs like Prolog and Datalog, and provide no efficient means of communication among parallel processes. Even if they are implemented in more modern parallel logic programming languages such as GHC and Parlog, the system is still *closed* in the sense that the program that represents multiple processes has to be written in advance and no change can be made to it after it starts execution. What we really need is a client-server model that enables communication and synchronization among *distributed* and *autonomous* agents (the clients) through a KBS (the server). The design of this new programming paradigm needs

to have the following properties: (1) it is an open system with a client-server architecture; (2) the server contains KBS as facts and rules and controls concurrency; (3) the client language extends the host language with a minimum set of communication primitives; (4) the primitives are expressive enough for clients to query, update and synchronize with the KBS, as well as coordinate among each other in both short and long transactions; and (5) the system is efficient and robust.

The OCP framework generalizes the Linda-style tuple space to a knowledge base, or a *constraint logic store*. An OCP query, known as a *reactor*, is a language neutral program fragment embedded in the host language. The agent sends the reactor to the OCP server and blocks until the reactor is processed and returned. Our reactor language includes a minimum set of expressive language constructs, including most notably the `sustain` construct. `sustain` allows the agent to do a sequence of actions as long as some logical condition is true. The sequence of actions is suspended if the condition becomes false and is resumed when the condition becomes true again. The `sustain` operation can be considered as a relaxed form of a transaction, because in many occasions, data consistency is safe guarded so long as the condition holds, and strict atomicity or serializability is not required. `sustain` provides the basic support for synchronization and coordination in the OCP paradigm.

To scale the OCP system to large number of agents, an advanced *triggering model* [4] was proposed to handle the blocking and wakeup of reactors. This triggering framework is built on top of the abstraction of logical conditions and its implementation relies on an efficient constraint indexing mechanism. The new indexing structure for this purpose is my invention of the RC-tree [5]. Unlike most traditional spatial indexes which do preprocessing of objects to approximate them as rectangles or other polygons, RC-trees index objects directly and segment them dynamically only when there is need to distinguish them. The segmentation is coupled with a smart “domain reduction” technique which reduces the minimum bounding rectangles (MBRs) of the segmented objects according to the shape of the objects. RC-trees rebalance dynamically like kd-trees. Experiments show that RC-trees outperform many existing indexes particularly in search efficiency and query accuracy.

Another main thrust in the OCP project involves a study of non-deterministic choices. One problem with the traditional committed choice in concurrent languages is that the program cannot “speculate”, i.e. attempt multiple choices at the same time, especially when updates are involved in these choices. We can show that speculation with interaction is useful in marketplace examples to help resolve deadlocks among agents and increase possibility of successful transactions. We added speculations into the OCP framework by introducing the notion of a generalized committed choice (GCC) [6]. GCC allows a program to execute (including updates to the store) multiple choices at the same time and only explicitly commit to one choice at a later time, at which moment all other choices are eliminated as if they never happened. Each choice of a program interact with other program in a virtual world, and the entire system is represented by a “multi-world”. This goes significantly beyond the deep guards in concurrent constraint programming (CCP), the long lived transaction in Saga, transaction logic, or ECA rules in active databases. The challenge in GCC is an efficient implementation that handles the explosion of multiple worlds. I developed a prototype GCC system that simulates a general producer-consumer scenario using a number of space/time controlling techniques.

I also built a prototype OCP server in C++ that is architected to handle open, simultaneous requests. At the heart of the server is a CLP(R) system as a KBS and a medium for synchronization. I also developed a Python library for embedding reactors in Python programs. With an efficient RC-tree index and trigger mechanism, the server is capable of handling over 1000 reactors per second on an Intel Pentium 4 Linux machine. In conclusion, this prototype system has achieved the open design, performance and flexibility needed in modern applications.

## Future Directions

Like it or not, ad hoc, semi-structured data is here to stay and despite progress, many challenges involved will remain. In the coming years, I look forward to investigating the following key problems:

*Very large data sources.* My long term goal is to develop an integrated distributed ad hoc data processing engine. The GLOVES system has taken a first step. However, both LEARNPADS and GLOVES have assumed that data is small enough to fit completely in the memory. This is certainly not true as we move toward very large-scale data sources faced by many real-world applications. Therefore, we need to revolutionize the way data is retrieved, archived, indexed, represented and learned. Some of the essential sub-problems include: secondary memory representation of ad hoc data; indexing of data both in-memory and on-disk; incremental retrieval of data on the network; distribution and integration of data processing tasks; and online format inference which evolves descriptions to match the evolving data.

*Interactive learning.* In general, it is difficult for computer-generated descriptions to match up the quality of the descriptions written by human experts who are better at picking tokens and taking semantics into account when making decisions about structuring. Hence, it would be useful to develop a system that includes the human into the loop. In our existing system, the refinement procedure is completely automatic and guided by MDL scores only. It is possible to make this phase an interactive process with an interface that allows the human to easily intervene. The human intervention also helps the local search procedure to escape from local optima.

*Light-weight processing.* I have started the design and the development of a language called PAWK, which shares many features and even syntax with the popular light-weight data processing language Awk. The difference with PAWK is that instead of querying data with regular expression, it represents data using PADS types and selects parts of data using an XPATH like path into the type structure. This offers a “quick-and-dirty” way of manipulating ad hoc data especially when the data is too complex to be expressed in regular expressions.

*Probabilistic modeling.* I would like to work with AI experts to investigate more advanced statistical models and machine learning techniques to solve the format inference problem. One major finding in probabilistic tokenization is that the quality and the distribution of training samples significantly influence the results. Selecting the best set of training samples and labeling them properly is almost essential to the success of these machine learning approaches. We need a way to gather large amount of ad hoc data sources for training purposes. One possible solution is to build an online format inference portal that helps people generate tools from their sample data and at the same time improve our statistical models using their data.

On the front of knowledge bases, I am prepared to investigate several new topics. First, I would like to work on the semantics and implementation techniques for speculation. The goal is to develop an open KBS that fully supports speculation. On top of this system, I would like to build a number of applications such as automated agent-based marketplace, air traffic control simulation system with collaboration from industry partners. I would like to further generalize this open system to include distributed KBS, that is, a constraint logic store distributed across many sites, with a first step taken in [3]. Distributed KBS is necessary as it is commonplace for large applications to rely on knowledge across many domains and located at different sites. It is also a desired property of any distributed system to have data replications across the network to increase fault tolerance. The distribution of the logic and facts poses extra challenge to the program control and the data consistency.

## References

- [1] K. Fisher, D. Walker, and K. Q. Zhu. LearnPADS: Automatic tool generation from ad hoc data. In *SIGMOD*, 2008.
- [2] K. Fisher, D. Walker, K. Q. Zhu, and P. White. From dirt to shovels: Fully automatic tool generation from ad hoc data. In *ACM Symposium on Principles of Programming Languages*, 2008.
- [3] J. Jaffar, A. E. Santosa, R. H. Yap, and K. Q. Zhu. Scalable distributed depth-first search with greedy work stealing. In *Proceedings of 6th International Conference on Tools with Artificial Intelligence (ICTAI)*, 2004.
- [4] J. Jaffar, R. H. Yap, and K. Q. Zhu. Coordination of many agents. In *Proceedings of the 21st International Conference on Logic Programming (ICLP)*, pages 98–112. IEEE, 2005.
- [5] J. Jaffar, R. H. Yap, and K. Q. Zhu. Indexing of dynamic abstract regions. In *Proceedings of the 22nd Intl Conf on Data Engineering (ICDE)*. IEEE, 2006.
- [6] J. Jaffar, R. H. Yap, and K. Q. Zhu. Generalized committed choice. In *Proceedings of the 9th International Conference on Coordination Models and Languages (COORDINATION)*, pages 191–210, 2007.
- [7] PADS project. <http://www.padsproj.org/>, 2007.
- [8] Q. Xi, K. Fisher, D. Walker, and K. Q. Zhu. Ad hoc data and the token ambiguity problem. In *Proceedings of 11th International Symposium on Practical Aspects of Declarative Languages (PADL’09)*, 2009.
- [9] K. Q. Zhu. *Open Constraint Programming, PhD Thesis*. National University of Singapore, 2005.
- [10] K. Q. Zhu, D. S. Dantas, K. Fisher, L. Jia, Y. Mandelbaum, V. Pai, and D. Walker. Language support for processing distributed ad hoc data. Technical Report TR-826-08, submitted for publication, Princeton University, July 2008.