robust distributed systems
achieving self-management through inference

Willem de Bruijn
Vrije Universiteit Amsterdam
wdb@few.vu.nl

Herbert Bos
Vrije Universiteit Amsterdam
herbertb@cs.vu.nl

Henri Bal
Vrije Universiteit Amsterdam
bal@cs.vu.nl

***Abstract*— Self-management has often been proposed as a means to reduce the growing complexity of administration in distributed systems. We argue that this can be achieved through aggressive automation of management tasks. To reach a high level of automation we propose to take an inference-based approach: codify best practices so that they can be reasoned about and adapted at runtime. Concerns specific to distributed systems are dealt with by the innate support for knowledge sharing.**

We introduce the methodology along with a reference architecture. The method's validity is tested by applying a preliminary implementation to a handful of practical problems.

I. INTRODUCTION

Computer networks interconnect a growing number of increasingly heterogeneous devices. Correspondingly, administration complexity is rising, leading to a reduction in overall system stability and an increase in maintenance cost. Management actions and subtle changes in the underlying infrastructure can have potentially widespread, unforeseen consequences.

To increase resilience we need to decrease dependence on human intervention. Not only is manual labor costly, recent figures show that it is the major cause of security related issues in enterprise networks [1]. Also, humans are increasingly becoming the performance bottleneck in the system, for instance when dealing with worm containment.

We propose an integrated approach to management of distributed resources: reducing *perceived* administration complexity by automating resource management tasks where possible. The ultimate goal is to make manual management disappear completely. In contrast to many other projects, however, our approach is decidedly bottom-up. By way of incremental steps (i.e., automating existing tasks) and support for composing complex tasks out of reusable smaller ones, the system becomes more self-organized. Adaptation occurs through runtime task optimization, based on up-to-date knowledge of the environment.

The precise actions to take depend on application-specific constraints, resource scarcity and interdependence of processes. For this reason a decision-making process must have detailed knowledge of the runtime environment, the submitted jobs and the administrative policies. Management tasks are generally well understood and decomposable, therefore they lend themselves well to automation.

We here propose to take an inference-based approach to handling complex management tasks. We argue that this approach has advantages over other efforts in self-management. In particular, it scores well on the following desirable properties: understandability, reusability, interoperability and extensibility. While the popularity of inference and knowledge-based approaches to network management is increasing, to our knowledge this is the first project that attempts to achieve these goals at this level. We describe the architecture and a first prototype.

II. METHODOLOGY

To be able to remove the human from the loop we must transfer his domain knowledge to an automated environment. Traditionally, micro-management tasks have been automated by ad-hoc shell scripts. While scripting can aid in solving simple problems, it falls short in the face of increased complexity as imperative programs encode relatively static procedures. Adaptation to new environments needs manual intervention. What we propose can be seen as a next step in scripting: *employing expert systems to adapt task handling code automatically*.

We will show that expert systems can be instructed to carry out monitoring and micro-management tasks more easily than scripts. Additionally, formal representation may help increase code and domain knowledge reuse. Especially of interest in distributed systems is the ease with which knowledge can be shared. Explicit knowledge exchange between partners in the distributed environment can help reduce work duplication and increase effectiveness in dealing with many issues from QoS negotiation to worm containment.

Adaptation can be achieved through a number of technologies, e.g., emergent behavior, Bayesian networks or hardwired parametric adaptation. For management tasks, expert systems hold some advantages over other methods:

1) **understandability** A line of reasoning, by virtue of its logical representation, can be easily followed and understood. This greatly benefits product development through reduced debugging time. Also, for this reason expert systems are less controversial than other methods of introducing self-adaptiveness.
2) **reusability** As explicit statements can be combined to create more (correlated) information, the total knowledge is greater than the sum of its parts (which is not true if knowledge is embedded and thus highly fragmented).
3) **interoperability** Translation between two formal languages is mostly trivial, while making two ad-hoc systems interoperate tends to be very cumbersome indeed.

4) **extensibility** High-level languages make knowledge engineering straightforward, which in turn encourages knowledge modification and extension by end-users.

III. ARCHITECTURE

Principal to our view are pervasive, knowledgeable local experts acting on behalf of stakeholders in the network. The task at hand, then, can be subdivided into two parts: (1) the construction and instruction of individual experts and (2) the coordination of communication between the actors in the network. By focusing on practical management challenges we will pursue a bottom-up, hands-on approach. Building a prototype local expert has our first priority. Increasing knowledge engineering return-on-investment through information exchange and reuse will be addressed in the second leg of our research.

A. local experts

The central element in our management environment is the local expert. This application receives runtime statistics from the network middleware and policies from its (human) controllers. Low-level runtime data is extracted by refactoring or mirroring existing monitoring tools such as SNMP or Tivoli while higher-level policies can be entered in a formal syntax or indirectly through friendlier (web) interfaces. Much research has already been undertaken into policy based network and grid management [2], [3].

Locally, we employ a model-based adaptation approach, similar to Muscettola [4] and Garland [5], but more flexible as the internal rules of an expert system can be scrutinized themselves. An expert builds up a model of its environment by correlating runtime system information. Through deduction he can draw more meaningful conclusions from these base facts. Depending on the given policies he may then choose to intervene directly, thus creating a closed-loop system or "self-managing habitat". Alternatively, he can opt for the less intrusive mode of solely reporting high-level information to his controller.

Many practical expert systems in use today have been built on top of predicate logic parsers. A major drawback of the dichotomous (true/false) nature of predicate logic is that it suffers from an incomplete representation of reality. In truth, we often have too little information to draw a conclusion with certainty. Various types of uncertainty can be discerned and even more methods to address them [6]. But unfortunately there is no silver bullet. We employ the most widely-used approach, rule-based reasoning, as it is flexible enough to accommodate for specialized methods where necessary.

The main challenges are (1) to define a language suitable for encoding policies and building a model of the environment, (2) to build an inference engine to work with the language and (3) to connect this expert to the network fabric through off-the-shelf tools.

B. global partners

Contrary to initiatives such as *zeroconf*, ad-hoc and sensor nets, we explicitly target enterprise networks and the internet, as our knowledge-based approach will ease communication and cooperation among widely dispersed actors. A pressing concern, then, involves scaling the presented local solution to potentially millions of interconnected devices. The distributed architecture poses additional challenges: where in the networks should we position the experts? How many experts do we need? How can they communicate safely and practically? How can we maintain such a distributed system?

Research in this field is active. On the one hand, we have systems-oriented research initiatives to implement the knowledge plane [7] vision of a global information space. On the other, we are beginning to see results from the knowledge representation society's efforts in building global knowledge-exchange languages, notably the semantic web [8] technologies. Building on these are plans for next generation integrated fabrics, among which are knowledge-oriented [9] and semantically driven grids [10] and virtual organizations of autonomous agents [11]. While these are exciting endeavors, our project attacks the problem from another, bottom-up, angle by expanding upwards from the individual expert.

Not only is this a pragmatic approach, we also feel it may be complementary to the aforementioned initiatives as the knowledge that we build up may subsequently serve as core concepts for any of the other approaches.

Regarding expert placement we note the following: centralized, or hierarchical approaches to distributing control are ill-suited to the internet as it inherently lacks centralized control (notwithstanding archaic tools such as DNS). Administrative boundaries are becoming increasingly vague, with groups of actors temporarily pooling their resources (p2p), individuals using many different resources at once (ubiquitous and pervasive computing) and systems becoming ever more interwoven (grids).

While knowledge may be seen as globally true, we expect optimization strategies to be inherently biased. Also, third-party information may be unreliable, intentionally or not. Therefore experts in a global space are to be *user-centric* and *skeptical*: able to fulfill their goals independently, although possibly through cooperation, with knowledge that is not 100% reliable. Thus, while not focusing directly on competing and cooperating autonomous entities (e.g., 'agents'), conceptually our work fits nicely within the view of virtual organizations.

IV. IMPLEMENTATION

Both as proof-of-concept as well as to learn how we can best codify management knowledge we have built a prototype expert system based on the previously discussed architecture: BetaGIS[1].

Expert systems have been proposed for wide-area automation before in the vision for a semantic web [8]. While most knowledge engineers, especially those working within the semantic web, directly expand their knowledge-bases through forward-chaining, we chose a goal-directed, or backward-chaining approach to inference. The most decisive factor was

[1]More information can be found at http://www.few.vu.nl/∼wdb/betagis

```
implements(TOOL, compiler),
accepts(TOOL, IN), mime(IN, text/x-csrc),
produces(TOOL, OUT), ext(OUT,'.x86.o'),
available(TOOL, HOST),
node_status(HOST,up).
```

Fig. 1. example query for a C to x86 compiler

our reliance on real-time data, such as remote sensors for CPU load or network availability, which makes precomputing correlated facts largely a wasted effort. BetaGIS was built on top of SWI-Prolog, which already has support for many of the features we require, such as an HTTP server and Semantic Web language handlers.

As our problem space encompasses many issues and Prolog is a versatile language we took care not to end up with a chaotic bag of unrelated code. Indeed, in order to cater to a wide range of problems with minimal code overlap we are continuously refining a base set of primitives. It is precisely the reusability of these primitives and the subsequent correlation of relatively disjunct knowledge that breeds potential adaptation (and may in addition serve as important input to other projects). As is explained next, the primitives found so far can be divided into two groups: relationships and tasks.

A. relationships

Relationships interconnect concepts and their instances. As a practical example, let us assume that an expert stores knowledge about a C compiler. Useful things to know are what sort of input it requires (gcc **accepts** C sourcecode), what sort of output it produces (gcc **produces** x86 object code), on what machines it is available, whether the machine is up, etc. Next, when implementing (for instance) a parallelizing version of `make` an expert can automatically find appropriate nodes on which to compile.

For this reason, the relationships that we implement cover not just the well-known `InstanceOf` and `ChildOf` relationships from description logic, but also less generic relationships, such as those concerned with ordering: `needs`, `wants` and the already introduced `accepts` and `produces`. Other relationships that have proven useful are `implements` for task to tool translation (gcc **implements** a compiler) and `available` for retrieving realtime information (gcc is **available** at node X).

As an example, the snippet of pseudocode in Figure 1 searches for a C to x86 compiler. Note that in Prolog words starting with capitals are free terms (variables), while all others have been bound. In the example we are trying to find such values for `TOOL` and `HOST` that all given relations evaluate to true. The first line searches for a tool that implements a compiler. The next two lines constrain the type of compiler: it must accept files of mimetype text/x-csrc and return files having the filename extension '.x86.o'. Then, the last two lines select a suitable location for execution: a reachable host at which the requested tool is available.

The example shows a drawback of predicate logic parsers: they only search for a correct answer, not an optimal one. However, such shortcomings can be circumvented. The example could find an optimum by selecting the node with the most free cycles.

B. task-handling framework

The second part of our core concepts covers not passive relationships but active processes. For this work we were able to build upon earlier research into the design of a flexible processing framework (FFPF [12]). Process descriptions are fairly simple: they detail what input, output and options processes require or accept. We use Prolog predicates to encode these statements. There are two types of descriptions: those encoding atomic tasks and those encoding composite ones. Atomic tasks embed actions such as running an executable. They are trivially encoded, e.g., `task(name,input,output,options).` Composite tasks can be built by combining other (possibly composite) tasks through explicit operators for sequential and parallel execution (resp. `seq` and `par`), similar to Occam [13]. This model of interrelated definitions supports the creation of an archive of task blueprints. As the prolog engine can 'understand' these blueprints it can slightly alter them to adjust to new circumstances. The precise number of parallel compilations, for instance, is easily adapted from a general parallelization pattern.

The basic building blocks of the framework are atomic tasks: descriptions of individual actions through which BetaGIS can interact with its environment. Through these it can utilize the large set of available (management) tools, such as `grep`, `net-snmp` or `procfs`, just like *human* experts. We rely on templates to create appropriate commands on-the-fly. The following simplified code snippet shows a template for the UNIX `diff` command. It contains the name, the command template and room for input, output and options, similar to a task. The last argument, written on the second line, is a constraint set. In this case, the input must be a list of two elements and the output a single element.

```
template(diff,'diff $opt $in1 $in2 > $out1',
In,Out,_,( length(In,2),atomic(Out) ))
```

Templates are matched with task requests to find a suitable fully bounded solution. As binding free terms is a two-way process in Prolog, both users and resources can constrain the option space. Let's reinspect the example in Figure 1. The `accepts` and `produces` relationships can be embedded in the following gcc template:

```
template(gcc, 'gcc $opt -o $out $in', In, Out,
'-m386',(mime(In, text/c-csrc),ext(Out,'.x86.o'))
```

This template will for instance combine with task-request

```
task(compiler, Cmd,'a.c','a.x86.o',_)
```

into

```
task(compiler,'gcc -m386 -o out.x86.o in.c',
'a.c','a.x86.o','-m386')
```

Supplementing these adaptable script snippets are predicates for specifying recurring tasks and for event handling. These

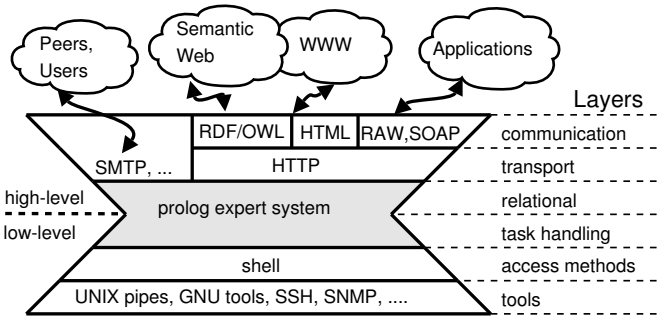

Fig. 2. local expert design

are used in background tasks, such as monitoring and maintenance. Together, these concepts are already powerful enough to encode many day-to-day management tasks. However, if needed we may extend them with other language constructs, such as flow-control (loops, switches) or even support a more formally grounded coordination language like Manifold [14].

At run-time, task specifications are translated into appropriate 'static' scripts. We can currently create shell scripts and visual flowgraphs. A drawback of this scripted approach is a lack of runtime feedback and consequent loss of control. On the other hand, removing the inference engine from the runtime system makes the process more easily traceable.

The maturity of the task-handling framework is best demonstrated by the fact that the expert system's own tasks are encoded in it. For instance, an automated knowledge-acquisition task is used to recursively build up knowledge about the network: foreign devices, their access methods and resources (tools, data and hardware) are queried in a `configure`-like manner. New information, e.g., obtained from the hosts file, can lead to more inspections, theoretically *ad infinitum*, while stale data is overwritten or removed. This process is written down using the task-handling code so that it automatically adjusts to new locations.

C. interfaces

While BetaGIS can directly control its environment by executing tasks, this mechanism isn't sufficient for interacting with end-users and peers. For that purpose it will need a more high-level method of communication. The SWI-Prolog shell is not sufficient for a myriad of reasons, among which are concerns about ease of use and automation. Instead, BetaGIS has an built-in HTTP client and server. As shown in Figure 2, the HTTP server supports three different interfaces.

The simplest interface accepts raw prolog queries and generates responses wrapped in a minimal envelope. As such, it allows BetaGIS to be asked questions from remote locations. Through command line HTTP requests (e.g. with `wget`) the interface even allows the embedding of BetaGIS actions into scripts. For instance, a host can add itself to the network at boot-time by executing the following shell script.

```
wget ${GISURL}/query?query=configure\($HOSTNAME\)
```

Entire toolchains can be replaced with their remote cousins in this fashion. For example, a `gcc` call can be intercepted and redirected to BetaGIS without the user knowing that other resources will actually carry out the request.

The second interface concerns interaction with end-users. This BetaGIS portal can help less tech-savvy users specify tasks or information requests in an easy-to-grasp graphical manner. Difficult queries are embedded in opaque HTML links that can be bookmarked for later use. Wizard-like interfaces encourage step-by-step knowledge acquisition, for instance for composite task specification. For monitoring purposes, the interface has also been given plot generation support. It visualized data in the manner of IBM's Tivoli or its open-source cousin Nagios, but as yet less refined.

A currently unused third interface allows communication with Semantic Web applications through RDFS+OWL language parsing. Many of our concepts are not compatible with RDFS or OWL at the moment, but we are contemplating changing this after the dictionary stabilizes.

Future extensions already shown in Figure 2 but not yet implemented concern SMTP for sending alerts and SOAP for integrating our task-handling framework with Web Services.

V. CASE STUDIES

BetaGIS is, as its name implies, first and foremost a research vehicle. Through its deployment we hope to find out which concepts are useful, even central, to management tasks and how we can optimally make use of these. More specifically we mean to attack the challenges posed at the end of Section III-A. To do so we also need ammunition in the sense of practical problems. So far we've implemented solutions for a small but diverse set of problems that we believe indicate the potential of expert systems such as BetaGIS.

A. end-user experiences

Environment personalization is a building block of ubiquitous computing. Our formalized tasks can help achieve this goal. Let's take, for instance, the mundane task of printing. In general a user wants to print a file at the nearest printer, preferably without having to specify the device explicitly (he may not even know its name).

Within BetaGIS, the location of an object can be specified using an extensible set of predicates. At the moment locations on the Vrije Universiteit campus, zipcodes and URLs are understood. These can be correlated, as in `vuloc(FEW,'R5.23')` is `near` to `zipcode(nl('1081HV'))`. At runtime, the `near` relationship and its cousin `nearest` can then be used to send the job to the printer nearest to the machine I am using. And to tell me where to fetch my print-outs, ofcourse.

B. distributed compilation

One highly decomposable task that has been the target of parallelization before is the `make` process. Tools such as DistCC and Prom [15] distribute compilation subtasks. But they do not address multistep compilations (`yacc`) or tool selection (`icc` vs `gcc`). BetaGIS *can* search for the optimal

toolchain at runtime. Relationships such as `produces` and `needs` construct a composite task, while `implements` and `available` ensure the task can be handled by the system.

C. monitoring OpenPBS

As a prototype of an advanced management task, we have almost completed the implementation of a fault recognition and recovery mechanism for the Portable Batch Scheduler (PBS), a set of network daemons controlling high-performance clusters. PBS has been found quite brittle. As not just the hardware, but also the OpenPBS software itself can fail at any time, we built background health sensors, together with event-handlers for known error conditions. Errors and their respective recovery mechanisms were identified through traditional knowledge engineering techniques, such as interviews. Consequently they are direct copies of the administrator's manual actions (e.g., logfile inspection, job cancellation and server rebooting). BetaGIS's ability to use standard shell tools helped reduce development time and program complexity for this scenario.

D. BetaGIS internals

The quintessential test for a piece of control software is whether it can monitor and control itself. BetaGIS uses its task-handling code for many of its internal actions. The `configure` task discussed in Section IV-B is one. Another, the web front-end, relies on on-demand graph construction through external tools such as `dot` and `GNU graph` for its visualization methods. Image requests are automatically forwarded to these applications. When multiple locations carry the application an optimization technique is applied, such as load-balancing. Similar mechanisms exist for remotely executing tasks in general (e.g., through `sh, ssh, snmp, http`).

VI. Related work

Perry and Wolf first proposed modeling software as an architecture of interdependent processes [16]. Architecture-based adaptation [17], even task-oriented [18], has been suggested previously. But so far adaptation is limited to optimization of quantified values, which are often hard to define correctly. The same problem also plagues approaches based on Bayesian reasoning [19]. Solutions specific to distributed systems [20], [21] lack BetaGIS's ability to leverage existing off-the-shelf tools. Systems-oriented approaches have so far been mostly limited to parametric optimizations. Delphoi [22] gathers low-level data and relates it to higher-level queries, but as queries are hardwired at compile-time, they are limited in scope. MDS-2 [23] implements a distributed knowledge exchange but lacks the inference needed to handle more demanding requests.

VII. Conclusion

Distributed systems become more robust and easier to manage when they become less reliant on human intervention. Automation based on formal encoding of best practices plus knowledge-based adaptation can help close the loop. A key advantage of the chosen approach is its support for the existing hard- and software infrastructure.

Validating our method, a prototype expert system was shown to be able to handle example problems. We plan to extend our work by (1) incorporating resource bounds and access restrictions into the framework and (2) scaling the solution to wide-area networks, as discussed in Section III-B. At the same time, we will extend and formalize our language and undertake more elaborate testing.

Acknowledgments

This work was carried out in the context of the Virtual Laboratory for e-Science project (www.vl-e.nl). This project is supported by a BSIK grant from the Dutch Ministry of Education, Culture and Science (OC&W) and is part of the ICT innovation program of the Ministry of Economic Affairs (EZ).

References

[1] J. Mortleman, "Humans to blame for security breaches." `http://www.vnunet.com/news/1154153`. "April 7th, 2004".
[2] J. D. Moffett and M. S. Sloman, "Policy hierarchies for distributed system management," *IEEE JSAC*, vol. 11, 11 1993.
[3] K. Yang, A. Galis, and C. Todd., "A policy-based active grid management architecture," in *IEEE ICON'02*, August 2002.
[4] N. Muscettola, P. P. Nayak, B. Pell, and B. C. Williams, "Remote agent: To boldly go where no AI system has gone before," *Artificial Intelligence*, vol. 103, no. 1-2, pp. 5–47, 1998.
[5] D. Garlan and B. Schmerl, "Model-based adaptation for self-healing systems," in *WOSS '02*, pp. 27–32, ACM Press, 2002.
[6] A. A. Hopgood, *Intelligent Systems for Engineers and Scientists*. CRC Press, 2 ed., 2000.
[7] D. D. Clark, C. Partridge, J. C. Ramming, and J. T. Wroclawski, "A knowledge plane for the internet," in *SIGCOMM03*, ACM Press, 2003.
[8] T. Berners-Lee, J. Hendler, and O. Lassila, "The semantic web," *Scientific American*, 5 2001.
[9] M. Cannataro and D. Talia, "Semantics and knowledge grids: Building the next-generation grid.," *IEEE Intelligent Systems*, vol. 19, no. 1, pp. 56–63, 2004.
[10] N. R. J. David De Roure and N. R. Shadbolt, "The semantic grid: Past, present and future," *IEEE Proceedings*, 3 2005.
[11] I. Foster, C. Kesselman, and S. Tuecke, "The anatomy of the Grid: Enabling scalable virtual organizations," *LNCS*, vol. 2150, 2001.
[12] H. Bos, W. de Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis, "Ffpf: Fairly fast packet filters," in *Proceedings of OSDI'04*, 2004.
[13] INMOS, *The occam 2 Reference Manual*. Prentice-Hall, 1988.
[14] G. A. Papadopoulos and F. Arbab, "Coordination models and languages," in *761*, p. 55, ISSN 1386-369X: CWI, 31 1998.
[15] T. Kielmann, "prom: A flexible, prolog-based make tool," Tech. Rep. TI-4/91, Technical University Darmstadt, 1991.
[16] D. E. Perry and A. L. Wolf, "Foundations for the study of software architecture," *ACM SIGSOFT*, vol. 17, no. 4, pp. 40–52, 1992.
[17] P. Oreizy, M. M. Gorlick, R. N. Taylor, D. Heimbigner, G. Johnson, N. Medvidovic, A. Quilici, D. S. Rosenblum, and A. L. Wolf, "An architecture-based approach to self-adaptive software," *IEEE Intelligent Systems*, vol. 14, no. 3, pp. 54–62, 1999.
[18] D. Garlan, V. Poladian, B. Schmerl, and J. Sousa, "Task-based self-adaptation," in *WOSS'04*, 2004.
[19] J. M. Agosta and S. Crosby, "Network integrity by inference in distributed systems," in *NIPS03*, 2003.
[20] I. Georgiadis, J. Magee, and J. Kramer, "Self-organising software architectures for distributed systems," in *WOSS'02*, 2002.
[21] J. Dietrich, A. Kozlenkov, M. Schroeder, and G. Wagner, "Rule-based agents for the semantic web," *Journal on Electronic Commerce Research and Applications*, vol. 2, no. 4, pp. 323–38, 2003.
[22] J. Maassen, R. van Nieuwpoort, T. Kielmann, and K. Verstoep, "Middleware adaptation with the delphoi service," in *agridm2003*, 2003.
[23] K. Czajkowski, S. Fitzgerald, I. Foster, and C. Kesselman, "Grid information services for distributed resource sharing," in *HPDC01*, 2001.