Sufficient Correctness and Homeostasis in Open Resource Coalitions:
How Much Can You Trust Your Software System?

Mary Shaw
Institute for Software Research, International
School of Computer Science
Carnegie Mellon University
Pittsburgh Pa 15213
+1 412 268 2589
mary.shaw@cs.cmu.edu

SYNOPSIS

Proving system correctness is tough.
It can fail, or succeed, or just bluff.
When the parts come and go
You may never quite know…
Can you tell when just "good" is enough?

ABSTRACT

Widespread use of the Internet enables a new class of software architectures: dynamically formed, task-specific, coalitions of distributed autonomous resources. The resources may be information, calculation, communication, control, or services. Unlike traditional software systems, which are at least nominally under control of the developer, these coalitions are formed from independently managed network-based resources, and the creator of a coalition lacks direct control over the incorporated resources. Reasoning about these architectures will differ significantly from reasoning about traditional architectures because resource coalitions experience higher uncertainty about component behavior and lower connector reliability. The traditional notion of correctness will give way to an application-relative notion of *sufficient correctness* for the intended use, and the traditional a priori means of validating correctness will give way to architectural provisions for reacting to abnormal behavior through *software homeostasis*.

Keywords: Open resource coalition, sufficient correctness, software homeostasis, fault tolerance

1 INTRODUCTION

The Internet has become a transforming technology, providing information and communication to a community that is orders of magnitude larger than it could have been just a few years ago. The critical elements of this transformation are communication connectivity, cross-platform delivery, rich resource availability, intelligibility to people who are not computer specialists, and a base for commerce. This is changing the ways we can create systems from parts, especially suitable architectures for such systems.

The recent exponential expansion of the Internet, especially the cross-platform access enabled by the World-Wide Web, opens a fundamentally new set of software development opportunities. Unlike traditional closed-shop software developments in which the system is controlled by a single institution and the system architecture draws on the traditional vocabulary [8], we now have a setting in which computation and information processing can be carried out by dynamically-formed, task-specific, coalitions of distributed autonomous resources.

These resources provide information, calculation, communication, control and services. They are independently developed and independently supported; they may even be transient. They can be composed to carry out specific tasks selected by a user – but often the resources are not specifically aware of the way they are being used, or even whether they are being used. Moreover, a resource may be used in a way that is incidental or secondary to its primary purpose.

Proprietors of the resources have their own objectives and priorities, and the selection and composition of resources is likely to be done afresh for each task, as resources appear, change and disappear. As a result, cooperating groups of resources are better regarded as coalitions than as systems.

I previously explored general architectural considerations for these web-based coalitions [11]. Here I turn to two specific related questions:

- *Sufficient correctness:* How do the coalition architecture and the criticality of the application interact to establish the degree of confidence a user must have in the coalition's correctness?
- *Software homeostasis:* How can that degree of confidence be achieved by a combination of a priori validation and dynamic monitoring and adaptation?

2 OPEN RESOURCE COALITIONS

The information resources of the Internet include much more than the static structured information and reusable code that we usually think of as reusable components. The variety of resources includes:

- *Information:* unstructured text, formatted text, databases, live data feeds, images, maps, current status such as inventory
- *Calculation:* reusable software components, applications that can be invoked remotely
- *Communication:* messages, streaming media, synchronous communication, agent systems, alert/notification services
- *Control:* coordination for use of resources, registration and subscription services
- *Services:* secondary (processed) information, simulation, editorial selection, evaluation, responsive experts

These resources are not functions, or files, or objects, or filters. They can, of course, be modeled or implemented in those terms, but to do so misses their richness. The resources are often large and comprehensive, providing multiple services; they may be modified independently and without notice.

Composition of components in this setting is difficult. It's hard to determine what assumptions each component makes about its operating context, let alone whether a set of components will interoperate well (or at all) and whether their combined functionality is what you need. Results are often simply exhibited through a browser, so it is often difficult to use a result from one resource as input to another resource. Moreover, many useful resources cannot be smoothly integrated because they make incompatible interaction assumptions – for example, it's hard to integrate a component packaged to interact via remote procedure calls with a component packaged to interact via shared data in a proprietary representation.

It is not realistic to expect complete specifications of a resource [9], so techniques for creating and maintaining coalitions must work with partial information. I previously introduced the concept of *credential* to capture those properties that are established, along with the degree of confidence in that information. The coalition itself must include credentials that document expected results, including an envelope of normal operation. As a result, analysis should focus on fitness to task rather than on correctness

A coalition will have a kernel in which the developer identifies resources, specifies the contributions to be obtained from the resources, and defines how these intermediate contributions interact to yield the result of interest. The technical issues associated with creating resource coalitions include creating appropriate credentials, or partial specifications; developing techniques for integration and automatic creation of glue code; ways of responding to the transience and mutability of the resources; incorporation of agent negotiations and security; and provisions for electronic commerce [9].

This wealth of available resources is appearing at the same time as the user community is expanding, especially the segment of the community with little computing expertise. End users increasingly control their own computing directly rather than engaging software specialists. This disintermediation began with spreadsheets and continues with end-user adaptation and program generators. However, end-user control of computation is no longer limited to application generators and spreadsheets – users now have access to a wealth of resources that are accurate, reliable, useful, and expensive to varying degrees, and they need easy-to-use means of harnessing combinations of these resources to their individual needs. Unfortunately, current software resources are so fragile that it's hard for a layman or anyone else to validate the software adequately.

3 SUFFICIENT CORRECTNESS

The "gold standard" of program quality has long been functional correctness – a demonstration that the program will compute what its (formal) specification claims it will compute. When the scale of the software grows from individual programs to complex systems that provide continuing service, that gold standard is augmented with extra-functional requirements such as performance and reliability. The important aspects of system correctness are sometimes associated with the architecture [10], for example in the use of the ACID properties (atomicity, consistency, isolation, durability) to evaluate data-centered repositories.

In practice, however, most of the software that most of us use most of the time is not "correct" in any traditional sense. Nevertheless, most of us do get useful results from this software. Most people view this situation more with resignation than with alarm; consider, for example, this teaser for a recent news report

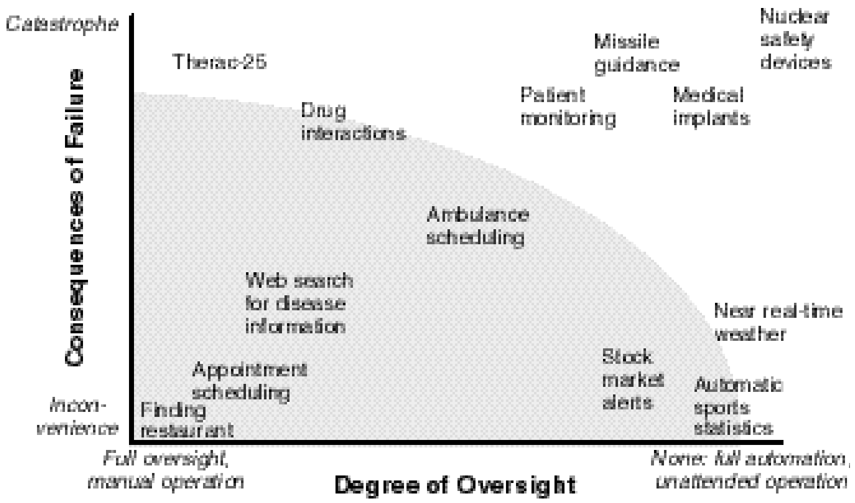
> Would you spend $500 on a piece of software with more than 63,000 potential known defects? That's the question posed in an internal Microsoft memo regarding Windows 2000 obtained by *Sm@rt Reseller*. According to the memo, Windows 2000, which makes its debut this week, contains: More than 21,000 "postponed" bugs, a number of which Microsoft calls "real problems;" more than 27,000 "BugBug" comments, which are ways to make something work better or more efficiently; and overall, more than 65,000 "potential issues" found using Microsoft Prefix Tool – 28,000 of which could likely be "real" problems. This news comes on the heels of many market research firms advising their clients to hold off on upgrading to Win2000 until the first or second service pack has been released. Microsoft contends that all software ships with bugs, and that Windows 2000 is the most rigorously tested software in history. [3]

This product is not alone in shipping with bugs; most commercial applications have flaws that can affect the correctness of their results, or indeed whether they deliver results at all. Networked resources are additionally subject to uncertainties of connectivity and communication.

Whether this situation represents potential inconvenience or dire risk depends a great deal on the kind of application and the environment in which it will be used. Certainly there are applications – medical implants and automated nuclear safety devices, for example – for which no less than the gold standard of correctness is acceptable. Just as surely, there are other applications – selecting a restaurant or a movie, for example – for which incomplete results or even failure are of little consequence.

By recognizing the many situations in which a reasonable chance of success (often combined with easily recognizable failure) is sufficient assurance for practical purposes, we open the possibility of applying forms of reasoning that are otherwise excluded, including statistical history, user recommendations, reputation of developers, etc. An example of the latter is acceptance of VeriSign [12] certificates; these authenticate the source but make no direct commitment about the quality

As a concrete example, if I were shopping for a new car, I would not hesitate to use a Web-based tool to search for a car that fits my needs. However, at present I absolutely would not entrust such a tool with US$20-30,000 and authorization to actually purchase the car. We need a way to decide whether a particular software system – for example, an open resource coalition – is sufficient trustworthy to be entrusted with the resources for a particular task.



Two factors affect our sense of the significance of less-than-correct behavior. First, we are usually less concerned about these failures when we are confident that we'll notice the problem and try again or perform some other remedial action. Second, we are usually less concerned about the failures when the consequence of the failure is small. As the figure suggests, these two factors define a space in which we can differentiate the significance of various sorts of failure.

I am interested here in the overall likelihood of acceptable system behavior– the end-to-end expectation of success. Moreover, I am interested in gaining assurance about fallible systems built from fallible parts. Improving the quality of the individual resources is an interesting question, but a different one.

The intrinsic uncertainties of resource coalitions, especially network performance and independent management of re-sources, make this architecture inappropriate for the critical systems in the upper right corner of the space. However, the shaded area contains many examples for which responsible risk management will find the cost of gaining full confidence higher than the cost of failure and repair. For such systems, we are interested in sufficient correctness: whether the system can be trusted to do what we intend to use it for. The result of the analysis should be an envelope of allowable behavior that is captured in the coalition's credential so that it sets the standard for initial and ongoing validation.

> ***Sufficient correctness:*** *The degree to which the system developer aspires to establish that the system meets its specifications, given constraints of time, cost, and limited knowledge.*

4 SOFTWARE HOMEOSTASIS

Changing the standard of quality from absolute correctness to sufficient correctness for the task at hand allows us to consider validation techniques that are too unreliable to use for absolute correctness. The architecture of resource coalitions provides a particularly good setting to explore these possibilities because of the intrinsic uncertainties of both the resources and the network connectivity.

I'm particularly interested in the strategic choices between prevention and repair. Generally speaking, there are two ways to deal with the possibility of undesirable behavior: taking advance measures to *prevent* it, or noticing that it has occurred and repairing the situation. The former is appropriate when the consequence of misbehavior is high, but it can introduce inflexibility, bureaucracy, and overhead. Nevertheless, classical correctness concerns and most software validation techniques are applied before the product is released, and so they fall firmly in the prevention camp.

In the case of resource coalitions, the independence of the resources means that they may well change while a coali-tion is running – so there is no alternative to considering the repair option as well as the prevention option. So in addition to introducing informal a priori validation methods, establishing assurances about resource coalitions requires the coalitions themselves to incorporate means of observing system behavior and reacting when it deviates from the allowable envelope, or even predicting when behavior is likely to depart the envelope.

> *Homeostasis is the propensity of a system to automatically restore its normal, or desired, or equilibrium state when something occurs to upset or disturb that state.* ***Software homeostasis*** *as a software system property refers to mechanisms for monitoring system behavior and dynamically modifying the system to repair deviations from expected behavior.*

Even after a coalition has been established and its adequacy established (e.g., it initially operates within the normal envelope), the constituent resources are subject to various classes of uncertainty, including

- ♦ Resource removed from network or moved to new URL by its proprietor
- ♦ Resource temporarily unavailable or performance poor because of network conditions
- ♦ Structure of resource changed substantially (e.g., directory structure of web site rearranged)
- ♦ Format of information delivered by resource changed
- ♦ Environmental assumptions of resource changed (e.g., needs upgrade of plugin)
- ♦ Protocol for accessing resource changed (e.g., registration now required)

Some of these uncertainties will immediately move the coalition's behavior out of the normal envelope; others may persist indefinitely; still others may be early harbingers of more serious problems.

The basic architecture of resource coalitions allows for four classes of internal mechanisms to detect the uncertainties:

- ♦ Exception mechanisms to intercept behavior outside the normal envelope by monitoring the communication channel
- ♦ Self-monitoring mechanisms that periodically run systematic checks on the resources. If semantic information is available in the credentials, this may include semantic consistency
- ♦ Statistical techniques based on machine learning or data mining to anticipate excursions from the envelope of normal behavior by analyzing historical data within the envelope.
- ♦ User-triggered detection

After an aberration is detected, the homeostatic mechanisms can act to repair the problems. Some actions can be taken silently and automatically:

- ♦ React to performance degradation by locating an alternate resource (e.g., a mirror) and switching the service. Go!zilla [1] is a stand-alone service with some of this functionality.
- ♦ Notice redirects, parse enough context to verify the presence of "we've moved" messages, and update the coalition's links
- ♦ Use an agent similar to Jango [2] or Computershopper [14] to select the most effective resource at each use
- ♦ Use a mediator such as the Typed Object Model server [6] to reconcile discrepancies

Other repairs should be submitted to the user for ratification or additional information:

- ♦ Notice unexpected requests for interaction (requests to download new upgrades or plugins, newly-introduced registration) and either follow up on the actions after approval or change the invocation protocol

Other problems may not be susceptible to automatic repair; in these cases the mechanisms can bring together enough information to make sensible error reports:

- ♦ Significant change in data format
- ♦ Completely dead links

Although this setting appears similar to adaptation in high-performance and high-assurance systems, the objective here is a bit different. Here the aspiration is satisfactory end-to-end performance from uncertain components; there it is high confidence for selected properties. This setting provides greater for informal reasoning.

5 EXAMPLE: NOW AND FUTURE

Suppose you operate a food concession at local fairs and entertainment events. You believe you can improve your profitability if you adjust the quantities of food you order based on weather forecasts, road construction reports, and amount of advance visibility in news reports. You can currently create a personal web page with links to web sites that provide relevant information, but you must gather and synthesize the information manually. Consider instead the possibility of creating an open resource coalition to monitor this dynamic information, synthesize the results, and notify you as appropriate. The tools for creating these coalitions (for example, extracting the information of interest from the result of an http: query) are subject of other research [11]. Here we consider homeostatic mechanisms that the construction tools might include in the coalition.

1. *Current and historical weather conditions:* Current weather is widely available from many sources, in many formats. For example, the Weather Underground [13] at http://www.wunderground.com/ provides current conditions and forecasts for many locations. The various weather sites regularly restructure their offerings, both by rearranging information on the city-specific report page and by reorganizing the page hierarchy. That is, a weather resource can fail by delivering information that the coalition can't interpret, or it can fail by reporting a nonexistent page. It can also fail transiently because of server or network problems, or it can be reporting stale or missing information because the weather resource's own sources failed. Some of these failures are transient and can be handled by waiting and retrying; some can be handled by switching to another weather resource whose organization and format is also known, and some require you to intervene (for example, when your primary weather site is restructured, you should figure out the new structure before the last of your alternate sites gets reorganized).

2 *Road construction reports:* Major construction projects don't change as much as the weather, but their current impact on traffic does. A general construction plan site such as http://www.realpittsburgh.com/partners/penndot/index.html [7] can tell you what projects are underway, but it may require project-specific information such as the construction schedule at http://www.libertytunnels.com/ [4] to let you know what will be happening on the day of your event. Sites like these are not updated as reliably as the weather, so your coalition should monitor updates and either warn you or seek out other sources if it is not updated.

3 *News archives:* Many newspapers maintain on-line archives; some require registration or subscription. For example, the New York Times operates a 365-day archive at http://archives.nytimes.com/archives/ [5]. This archive can be searched free, though full articles cost $2.50. You can, of course, search these sites manually, but you probably want to search several of them automatically and report statistics on the results. Your statistics may be misleading if some of the news sites don't reply to queries, so your coalition should monitor its response rates and adjust summary statistics accordingly.

6 RESEARCH OPPORTUNITIES

The Internet changes the usual assumptions about system composition in significant ways. Open resource coalitions offer a promising architecture for exploiting the new opportunities. In addition to technical challenges, this setting provides new challenges in setting expectations for system performance and for ensuring that these expectations are satisfied. I propose two research challenges: exploring criteria for sufficient correctness and implementing mechanisms for software homeostasis.

ACKNOWLEDGEMENTS

My understanding of Internet-based software systems as coalitions of independent resources has emerged by trying to reconcile what I thought I believed with what I saw around me. Discussions with colleagues in the Carnegie Mellon School of Computer Science, especially Orna Raz, David Garlan, Bill Scherlis, and other members of the Composable Software Group, laid the groundwork. Long ago, Vic Vyssotsky encouraged me to think about large-scale software development as an activity more akin to city planning than to product manufacture, and I'm gradually coming to terms with the implications of that advice.

REFERENCES

1. Aureate Media. Go!zilla Monster Downloads. http://www.gozilla.com/, February 2000.
2. Excite. Excite Product Finder, powered by Jango. http://www.jango.com, February, 2000.
3. Mary Jo Foley, Sm@rt Reseller. Bugfest! Win2000 has 63,000 'defects'. *ZDNet News*. Full article at http://www.zdnet.com/zdnn/stories/news/0,4586,2436920,00.html, February 2000.
4. The Liberty Tunnels Interchange Project. http://www.libertytunnels.com/, April 2000.
5. New York Times. 365-day Archive. http://archives.nytimes.com/archives/, February 2000.
6. John Ockerbloom. *Mediating Among Diverse Data Formats*. PhD Thesis, Computer Science Department, Carnegie Mellon University, 1998.
7. The Pittsburgh Road Report. http://www.realpittsburgh.com/partners/penndot/, April 2000.
8. Mary Shaw and David Garlan. Software Architecture: Perspectives on an Engineering Discipline. Prentice Hall, 1996.
9. Mary Shaw. Truth vs Knowledge: The Difference Between What a Component Does and What We **Know** It Does. *Proc. 8th International Workshop on Software Specification and Design*, March 1996.
10. Mary Shaw and Paul Clements. A Field Guide to Boxology: Preliminary Classification of Architectural Styles for Software Systems. *Proc COMPSAC '97: Int'l Computer Software & Applications Conference*, 1997, pp. 6-13.
11. Mary Shaw. Architectural Requirements for Computing with Coalitions of Resources. Position paper for *First Working IFIP Conference on Software Architecture,* http://www.cs.cmu.edu/~Vit/paper_abstracts/Shaw-Coalitions_paper.html, 1999.
12. Verisign. Internet Trust Services. http://www.verisign.com/, February 2000.
13. Weather Underground. Wunderground.com (current and forecast weather conditions) http://www.wunderground.com/
14. ZDNet. Computer Shopper. http://www.zdnet.com/computershopper/index1.html, February 2000.