Self-Healing Open Systems
(Position Paper)

Stephen Fickas\(^1\) and Robert J. Hall\(^2\)

\(^1\)Computer Science Department, University of Oregon
\(^2\)AT&T Labs Research, Florham Park, NJ

1. Introduction

Integration and interoperation have become the critical issues in engineering multi-stakeholder distributed systems (MSDS) like the Internet electronic mail system, networks of web services, modern telephone networks, and the Internet itself. Consistent, well defined protocols and other low level requirements enable these systems to function, but higher level requirements placed by diverse users are often ephemeral and typically inconsistent when viewed together. Thus, for the field of requirements engineering to deal with open MSDSs at all, we need to shift our thinking from systems having consistent, global requirements to those in which requirements can be user-relative and ephemeral.

Beyond that issue, however, lurks a second major challenge dubbed the "ignorance problem": since the nodes of an MSDS are controlled by stakeholders with different goals, priorities, and capabilities, just knowing what they all do is a challenge. For example, email features and functionality have grown so complex that merely knowing a host serves TCP port 25 (SMTP) does not give enough information to know whether one's email message will be handled correctly. Web services, while often packaged with interface information, are similarly opaque to formal or informal validation, often providing only brief and ambiguous natural language hints as to their functionality.

To make progress on this problem will require bringing together research in requirements engineering, component-based design, verification and validation, and related fields to discuss the challenges of designing and using open systems in which requirements are ephemeral and user-relative, and in which it is difficult or impossible to know the behaviors of all the parts of the system.

In the remainder of this position paper, we will layout the issues that we see as key, and provide a small example to motivate the research.

2. A more detailed characterization of the problem

The work of the two authors has turned up a set of issues for making progress on open systems [Fickas et al, 2002a,b][Hall, 2002]. We list these issues in bullet form to be brief.

- In an open system, upfront system analysis is at best of limited, heuristic usefulness. The system is amorphous and dynamic: it comprises whatever components are available at the moment. The best one might hope for is just-in-time analysis: can requirement R be met at this moment with system S?

- Two new types of requirements are prevalent in open systems:
  a. How do we capture and reason about ephemeral requirements? Ephemeral requirements have a lifetime that can be measured in days, hours, minutes or even less. They may be recurring, but may also be one-off.
  b. How do we capture and reason about user-relative requirements, or what we might call personal requirements? Personal requirements are not global to the system, but are specific to a single user or stakeholder. They are highly contextual.
• Requirements in open systems are often obstructed by the environment, i.e., by components not under the user's control. It is less interesting to know that a requirement R cannot be met than knowing how it might fail. Both runtime monitoring and failure-recovery mechanisms come to the fore.

• Given that many components of the system will not be under a user's control, how can we reason about the system's behaviors that may be relevant to the user? Can components provide an external description that will allow us to reason about requirement achievement? Can components be made transparent (e.g., reflective) so we can directly reason about their behavior?

• Can open modeling standards be established that would allow personal requirements validation tools to help a user discover and reason about personal requirements satisfaction? If so, how can communities of interest establish shared ontologies/theories that can support capabilities like semantically meaningful model composition, scenario simulation, animation, theorem proving, model checking, and other formal reasoning tools? What information (beyond the standard OO entity-relationship information) must be included in such shared theories?

3. An example

We will discuss the problem of email request and reply in an open system. This example is drawn from a real example [Fickas et al, 2002a]. User X wishes to get a ride to the doctor from user Y. User X decides to send an email request to Y asking for the ride. In slightly more formal terms, X's requirement is a yes or no answer from Y to the ride request, in time to do X some good. The requirement is ephemeral: it endures for this ride request only, and may never come up again. The requirement is personal: it is X's requirement and contextualized to the components in place, at the request moment, to get a question to and reply from Y.

It is an open system problem in that neither the Internet connectivity of X and Y, the communication channel (the SMTP server and post-office server) between X and Y, nor the email client of X and Y are under the control of all parties. In particular, X and/or Y can be disconnected, email servers on either X's or Y's side may filter email under administrative policy, the actual email clients themselves can filter email. Further, from X's point of view, Y (the human) is part of the environment: Y can ignore the request. In summary, the ride-request requirement can fail in many ways and it is impractical to attempt to guarantee success.

The questions this example raise in terms of the workshop are as follows:

• Can we reason at all about the potential success or failure of the ride request? Do we know or can we discover what system components must be involved in the request? If we know the components, can we also know their behavior? If we know their individual behavior, can we reason about success and failure from a system-wide level?

• Once we know how the request can fail, can we do more than hope for the best? Can we monitor its progress? Can we be warned of impending failure and potentially circumvent it? If failure does occur, can we recover in a way that keeps the request alive?

• Is it feasible, in a limited domain like email, to develop a shared ontology, one that all email server and client vendors might adopt?

• Privacy issues arise in this problem. It may be the case that Y's email client is willing to make explicit the general rules it follows, (e.g., email from certain domains on a watch-list are deleted) without providing details (e.g., whether X's domain is on the watch list).

4. Current Work

Both authors have made some fledgling attempts to come to grips with some of these problems. In particular, Fickas is using the email example from the last section as a test problem. His group has built tools that can capture and monitor requirements like the ride-request. This is discussed in detail in [Fickas et al, 2002b]. His focus is now on
the “self healing” aspect of the problem. Can runtime monitors give a warning in time to salvage the requirement, either by manipulating components under our control, or attempting to influence components that are part of the environment? Fickas has looked at the same issues in another example, that of an onboard spacecraft system attempting to maintain its functionality in an environment that can cause failure in several ways [Fickas et al, 2002a]. An interesting note from this work is that NASA has many manual fault-recovery procedures in place. The question is whether these can be transformed into automated, self-healing systems given that we can supply runtime requirements monitoring.

Hall has also looked at email systems as exemplars of open systems. In [Hall, 2000], he used a tool supported heuristic validation method to analyze combinations of email feature models to find undesirable behaviors and feature interactions. This work demonstrated the usefulness of component models in supporting validation of personal requirements in open systems. The study also motivated his more recent work on OpenModel [Hall, 2002], which attacks the ignorance problem by attempting to define an open modeling standard for MSDSs in a given domain. Key to the success of that work is understanding how to build and share domain specific ontological theories that support various sorts of automated reasoning tools, like simulators, animators, theorem provers, and model checkers. These shared theories must be negotiated and agreed upon by the diverse stakeholders within a domain, such as email, web service domains, or telephony systems.

For a system to be capable of “self-healing”, there must be a notion of “correct behavior”. As we have argued, open systems do not really have a single central notion of “correct”, as requirements must fundamentally be personal and ephemeral. Thus, we would argue that self-healing within open systems must be understood relative to these personal and ephemeral requirements. Hence, any self-healing open system must first have the capability to monitor and diagnose violations of these requirements. It must also have mechanisms for propagating and updating knowledge of node behaviors in order to support monitoring, diagnosis, and repair. We believe our current efforts are first steps on the road to self-healing open systems.

Acknowledgements

This work was partially supported by grants from the DARPA DASADA program, the Jet Propulsion Laboratory and from the National Institute on Disability and Rehabilitation Research (NIDRR). We gratefully acknowledge this support.

References


Fickas, S., Helm, R., Automating the design of composite systems, IEEE Transactions on Software Engineering, June, 1992

