What SOA can do for Software Dependability

Karl M. Göschka, Lorenz Froihofer, and Schahram Dustdar
Vienna University of Technology, Institute for Information Systems,
Distributed Systems Group, Argentinierstrasse 8/184-1,
A-1040 Vienna, Austria
{karl.goeschka|froihofer|dustdar}@tuwien.ac.at

Abstract

The prosperity and competitiveness of organizations and societies in general largely depend on the degree of their ability to react flexibly and pro-actively on a constantly changing environment. As many new products and services depend on information and communication software, the inertness of today's software systems turns them into an obstacle rather than an enabler and results in dependability degradation during the systems' lifetime. Even more so, heterogeneity, scale, and dynamics open up what Laprie called the "dependability gap".

In this position paper, we identify two research directions to close the dependability gap: First, to improve the integration of software systems with the business functions they support. Second, adaptive and evolvable systems based on the control loop approach. For both research directions, we will show how and to what extent SOA provides support today—and what is needed to foster the true potential of SOA for dependable and agile software systems tomorrow.

1. Service Science

The prosperity and competitiveness of organizations and societies in general largely depend on the degree of their ability to react flexibly and pro-actively on a constantly changing environment. As many new products and services depend on information and communication software today, we witness the trend towards Service-oriented computing which is mainly driven by industry and welcomed by many computer scientists. Service-Oriented Computing (SOC) – often called Service-Oriented Architecture (SOA)—is a computing paradigm that utilizes services as the basic constructs to support the development of rapid, low-cost and easy composition of distributed applications even in heterogeneous environments.

This trend has further led to a proposal called *Service science*, which suggests a tighter integration of disciplines including management science, computer science, operations research, industrial engineering, business strategy, social and cognitive sciences, and legal sciences to develop the skills required in a services-led economy. Service architects who are capable of planning, designing, and rolling-out business processes and their implementations as software services, are required to have an in-depth knowledge on human aspects, organizational aspects, business processes, and computational and software aspects. Complex services provide novel problems and pose new challenges for many disciplines, which ultimately leads to the aim of a better integration of business needs with technological solutions.

One contribution of this paper consequently is to provide a conceptual framework for a better integration of business needs and technology, which clearly identifies areas for future SOA research to foster business integration in Section 4. Before that, in the next section we describe current dependability problems in order to motivate our approach provided in Section 3. Then, in Section 3.3 we show what SOA can do for dependable systems today—and where it falls short.

2. Dependability

While computing is becoming a utility and software services increasingly pervade our daily lives, the integration of technology with business has to be complemented by technical approaches that turn software systems into an enabler of business agility, while providing the necessary dependability. Hence, dependability is no longer restricted to critical applications, but rather becomes a cornerstone of the information society. Dependability clearly is a holistic concept: Contributing factors are not only technical, but also social, cultural (corporate culture), psychological (perceived dependability), managerial and economical. Fostering learning is a key, and simplicity is generally an enabler for dependability.

Among technical factors, software development methods, tools, and techniques contribute to dependability, as defects in software products and services may lead to failure and also provide typical access for malicious attacks. In addition, there is a wide variety of fault tolerance techniques available, ranging from persistence provided by databases, replication, transaction monitors to reliable middleware with explicit control of quality of service properties and aspect orientation. Adaptiveness, self-properties, and autonomous computing are envisaged in order to respond to short-term changes of the system itself, the context, or the users' expectations.

Unfortunately, heterogeneous, large-scale, and dynamic software systems that typically run continuously often tend to become inert, brittle, and vulnerable after a while. The key problem is, that precisely the emerging systems and applications are the ones that suffer most from a significant decrease in dependability when compared to traditional critical systems, where dependability and security are fairly well understood as complementary concepts and a variety of proven methods and techniques is available today [2]. In accordance with Laprie [8] we call this effect the dependability gap, which is widened in front of us between demand and supply of dependability, and we can see this trend further fueled by an ever increasing cost pressure. This is partly caused by some of the following reasons [12, 8]:

- Change of context and user needs: It is impossible to reasonably predict all combinations of change during design, implementation, deployment, and—most importantly—during run-time.
- Imprecise (and sometimes even competing or contradictory) requirements: Users are either inarticulate about their precise criteria for correctness, performance, dependability, and other system qualities, or different users impose competing or contradictory requirements on the system, partially because of inconsistent needs.
- Interdependencies between systems and software artefacts, and emerging behaviour: The system may be too complex to predict even its internal behaviours precisely. Complexity theory [9] clearly shows that the overall properties of a complex software system are largely determined by the internal structure and interaction of its parts and less by the function of its individual constituents.

As a result, traditional systems experience permanent *dependability degradation* throughout their life-time. This in turn requires continuous and highly responsive human maintenance intervention and repetitive software development processes. While this need for intervention is costly, error-prone, and hence further impairs dependability, it may, in some cases, even become prohibitively slow compared to the system's pace in normal operation.

Hence, our second contribution is to suggest a (potentially nested) control loop approach that converges methods from software engineering with methods from traditional dependability research, as devised in the following section.

3. The control loop approach

Two complementary approaches address the problems of dependability degradation and the dependability gap: Adaptive coupling and run-time software engineering.

3.1. Adaptive coupling

First, large and dynamic systems can benefit from short-term adaptivity to react to observed, or act upon expected (temporary) changes of the context/environment (e.g., resource variability or failure scenarios) or users' needs (e.g., day/night setting). As this kind of adaptivity (also termed software agility) should be provided without explicit user intervention, it is also termed autonomous behavior or self-properties, and often involves monitoring, diagnosis (analysis, interpretation), and reconfiguration (repair) [5].

One of the main reasons why many approaches fell short in the past, lies in the major focus on the system's components (e.g., by focusing on recompilation, reconfiguration, and redeployment of components), while complexity theory [9] on the other hand clearly shows that the overall properties of large and complex software system are largely determined by the internal structure and interaction of its parts and less by the function of its individual components.

Service oriented architectures (SOA) already address this perception by putting a strong focus on structure completely separated from the implementation of individual constituents. So far so good, but there is still an important aspect missing: The internal structure of a system is formed by relationships of differing strengths between constituents. Components with tighter connections (or coupling) cluster to sub-systems, while other components may remain more loosely-coupled with each other or with clustered sub-systems. Hence, a complex software system provides a mixture of tightly and loosely coupled parts. As an important consequence, the overall system properties are determined not only by the structure but also by the strength of coupling of its relationships.

Thus the inner control loop has to adaptively control the strength of coupling between the system's constituents as the most promising approach to influence/control its overall dependability properties.

To provide the desirable degree of adaptivity, competing dependability and security properties of the overall system have to be explicitly balanced according to the re-

spective situation (context, failure scenarios, current user needs). This balancing should flexibly be performed as interaction between infrastructure and application (or even the end user) through the explicit control of adaptive coupling mechanisms between software services, typically through run-time selection and reconfiguration of dependability protocols, e.g., consistency of replication protocols.

3.2. Run-time software engineering

Second, as not all possible evolvements can be foreseen for long-running software, long-term evolution has to be supported to regulate the emerging behavior of large and dynamic systems, again, with respect to the evolvement of the requirements and user expectations, but also in response to long-term changes in the context.

This will be performed by changing the system's design during run-time, which in turn requires run-time processable requirements and design-views in the form of constraints [4], models ("UML virtual machine"), or (partial) architectural configurations. The ultimate idea here is to move into run-time what previously could only be done by modifying an application off-line during design-time.

These run-time accessible and processable requirements can be stored in repositories or be accessed via reflection, aspect-oriented programming, or protocols for meta-data exchange. They can explicitly be manipulated and configured, which allows such a system to balance or trade certain properties against each other or against user needs during run-time.

Clearly, this requires middleware services to support access and manipulation of requirements and negotiation of properties and needs. The vision here is a convergence of software development tools with middleware (including traditional dependability, fault tolerance, and adaptivity concepts), to provide for run-time software development tools to compensate for dependability degradation by re-engineering running software. Yet, in turn, this introduces new challenges for dependability engineering, requiring methods for run-time verification and testing, for example.

Figure 1 shows the outer control loop: The dependability properties of the system are measured ("software sensors") and compared to the users' current needs. During a negotiation process, it is decided which properties are traded against each other according to the current system state, context, and users' needs. Finally, the system is changed via adjustment of run-time processable requirements, in order to achieve the properties as intended. Short-term adaptivity and long-term evolution can accordingly be differentiated as a combination of two nested control loops, where the inner loop represents adaptivity and the outer loop (as depicted in Figure 1) represents software evolution.

Regardless of the pace of change, both approaches address the imprecise, emerging, and ever-changing nature of large and long-running software systems and introduce iterative steps of adaptation and evolvement during run-time. Both approaches are needed in practice and will need different solutions, but have in common the need for

1. reconfiguration of the architectural coupling and adaptive usage of differing strengths of communication coupling,
2. measurement of dependability properties in order to provide means to balance them and interactively negotiate them with users, and
3. run-time processable meta-data representing the current run-time structure and the design-view, including explicit context dependencies, explicit ordering and wiring, explicit redundancy control.

3.3 Service-Oriented Computing

Although one could—in principle—build a system as described above with almost any kind of technology, some are more efficient and beneficial than others. Service-Oriented Computing (SOC) is an emerging computing paradigm utilizing services to support the rapid development of distributed applications in heterogeneous environments. The visionary promise of service-oriented computing is a world of cooperating services being loosely coupled to flexibly create dynamic business processes and agile applications that may span organizations and computing platforms and can, nevertheless, adapt quickly and autonomously to changes of requirements or context. Web services may be dynamically aggregated, composed, and enacted as Web services workflows. In the following we investigate, how and to what extent SOC can currently support adaptive dependability, but also new challenges that arise from the deployment of SOC. From the most important standardsthe following relate to our work:

Web service coordination: The Web services coordination framework (WS-Coordination [7]) provides a foundation layer for consensus between Web services, where specific consensus protocols can be built upon, e.g., distributed transactions. A service-oriented transaction model has to provide comprehensive support for long-running transactions between loosely-coupled (federated) service partners and resources, including negotiations, conversations, commitments, contracts, tracking, payments, and exception handling, thereby contributing to the notion of adaptive coupling. Two particular standards build upon the WS-Coordination framework: The WS-AtomicTransaction [7] and the WS-BusinessActivity [1].

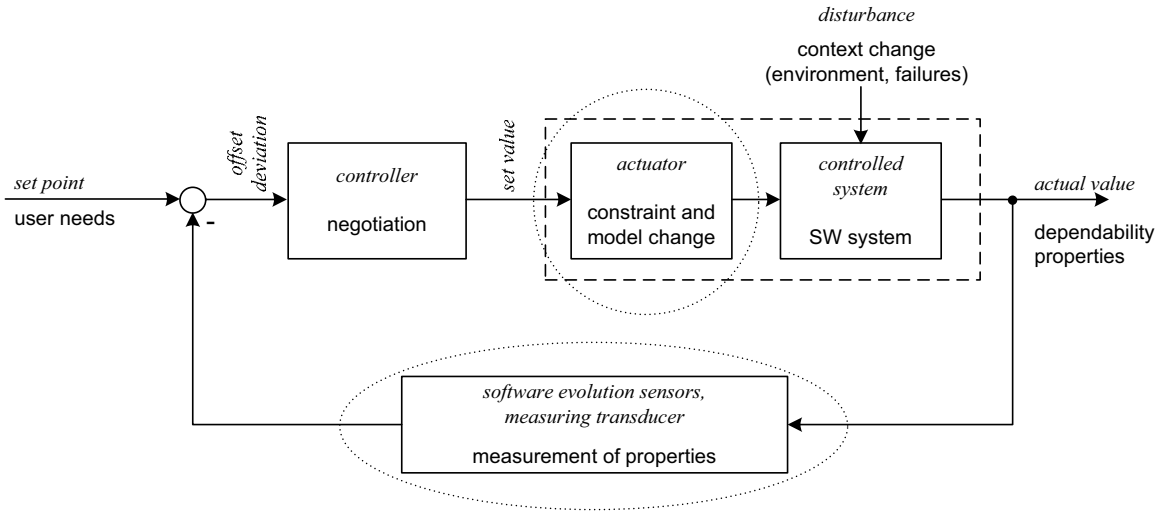

Figure 1. Control loop of dependable evolution through run-time model-/constraint-change.

Web service publishing and discovery. The Universal Description, Discovery and Integration (UDDI) protocol is one of the major building blocks for binding (and therefore coupling) of Web services, alternate approaches are available as well [3].

Web service meta-data exchange. This standard [6] explicitly addresses the need for extensive exchange of meta-data in an environment which deploys adaptive coupling. It provides a framework to support meta-data for particular purposes being exchanged between interested participants and is therefore an important means of run-time manipulation of software.

Service oriented middleware (SOM). The highly dynamic modularity and need for flexible integration of services lead to the question to what extent service-orientation at the middleware layer itself is beneficial (or not). While the idea is tempting, to provide services like a transaction service or a group membership service (service groups), in some cases such a "service" is more a system's aspect than a coherent service, like for instance most dependability attributes. Providing end-to-end dependability and autonomic capabilities in a heterogeneous, potentially cross-organizational SOA is a particular challenge and the limits and benefits have still to be investigated.

Service replication. [11] compares state-of-the-art service replication middleware with object replication middleware on an architectural level in the case of strict consistency: Object and service replication middleware share many commonalities and only subtle differences, caused by (i) the different granularity of objects and services and (ii) different technology standards (e.g., CORBA vs. WS). Clearly, the wheel need not be re-invented here.

Summing it up, service-oriented computing addresses some needs for adaptive dependability between a system's constituents (coordination, discovery, meta-data), and hence contributes to the dependability of such systems. On the other hand, SOC poses new research challenges, in particular for the SLA of typical end-to-end properties, like dependability *guarantees*. In some cases, SOC turns out to be "yet another technology", but does not contribute to improvements, which is, e.g., the case for replication. Therefore, SOC as it is today provides some promising aspects, but today's realizations still fall short to address the full scope of needs of adaptive, dependable systems. Hence, future research is needed to develop SOC standards and technology to their full potential.

4. A framework for business integration

In many cases, service-oriented architectures can now realize the vision of a "plug-and-play" business IT infrastructure. Web services could also leverage the creation of business networks through which aggregations of products and services can flow freely. They have the potential for transforming how businesses and organizations interact within themselves and with others. Web services are expected to fuel a new wave of electronic business, application integration, and business-to-business (B2B) interactions, as the industry moves towards application and service integration, rather than dedicated system development that require extensive design, deployment and integration efforts.

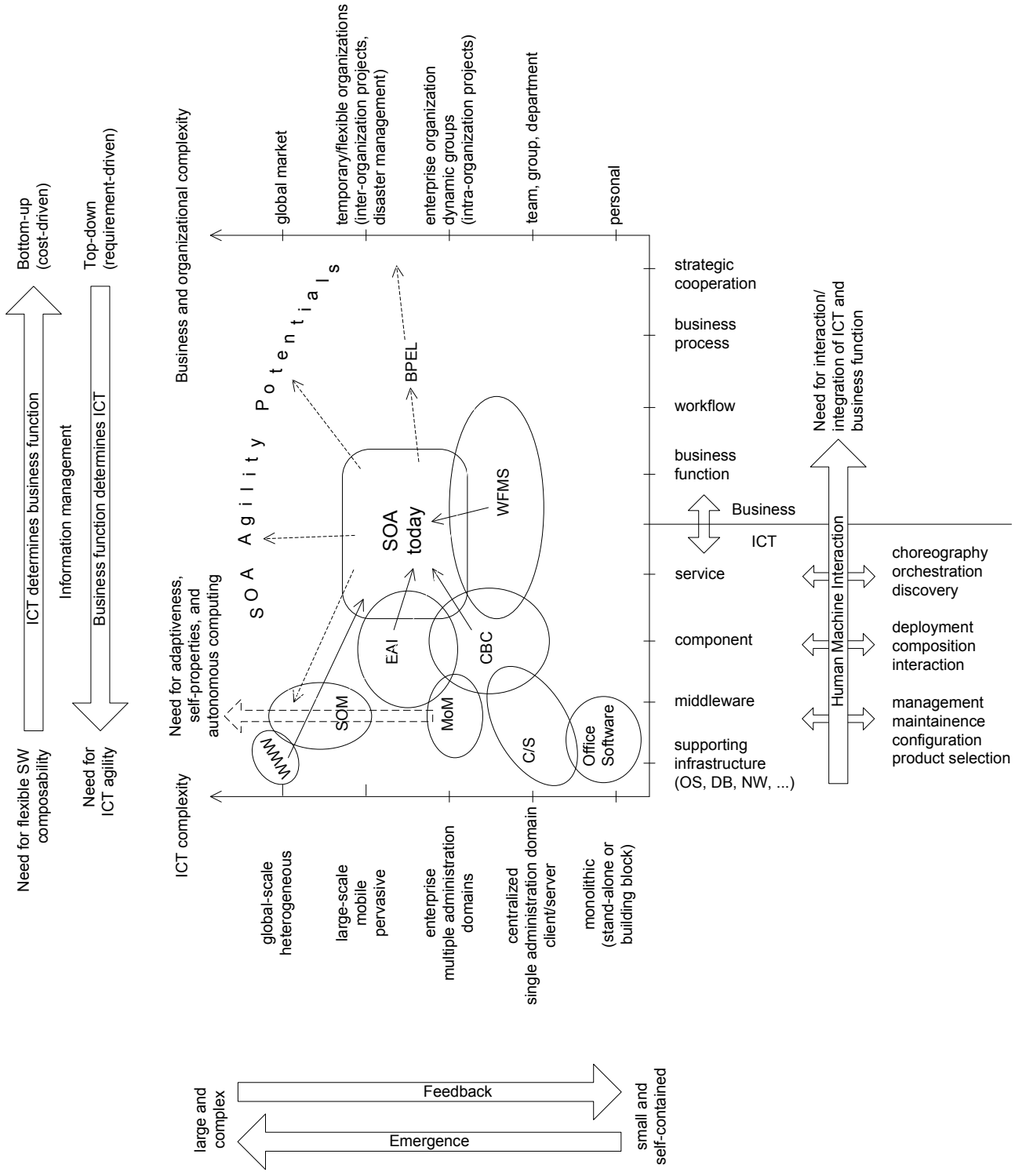

Figure 2. A framework for agile business integration.

Business literature as well as scientific literature (for a good overview on both aspects see e.g., [10]) discuss the tight dependencies between information systems (in particular based on software services) and business functions. Accordingly, Figure 2 integrates our findings into a framework for agile integration of business functions and software services: The y-axis denotes increasing complexity, on the left-hand side with respect to IT complexity and on the right-hand side with respect to business complexity. From simple to more complex entities the systems provide emerging behavior, while it is important not to forget the feedback from the large system onto the simple building blocks. The x-axis denotes the integration of IT and business, following a common layered approach. The bottom-up approach of software composition imposes the need for flexible composability, while the top-down approach of software requirements analysis demands agile software development processes and ICT (Information and Communication Technology) agility in general. Notably, human machine interactions with the IT building blocks take place at each of the layers following different tasks. So far we do not experience IT-based user interactions with the business layers, which is definitely in the scope of future research work.

The subject of Service Oriented Computing is vast and enormously complex, spanning many concepts and technologies that find their origins in diverse disciplines like Workflow Management Systems (WFMS), Component Based Computing (CBC), "classical" Web applications, and Enterprise Application Integration (EAI) including Message Oriented Middleware (MoM). This is shown in the center of the framework figure. In addition, there is a strong need to merge technology with an understanding of business processes and organizational structures, a combination of recognizing an enterprise's pain points and the potential solutions that can be applied to correct them. This is shown with the arrows denoted as "SOA agility potentials", pointing into deeper business integration or increased complexity. Notably, there is one emerging area dealing with increased complexity but with less business integration: The question of Service oriented Middleware (SOM), that is, to what extent middleware itself should be service-oriented.

Summing up, Figure 2 provides a convenient overview to explicitly address the two dimensions of complexity and business integration from the software (IT) point of view. It allows to arrange technologies accordingly to show their interrelations and correlations, and to precisely describe the required future research directions for dependable and adaptive software systems on different levels ranging from different software layers to business processes. It also stresses the fact that complexity and business integration are not just two separate aspects, but rather two dimensions of agility concepts in general.

5. Conclusion and future work

The success of businesses and organizations increasingly depends on their flexibility to adapt to a constantly changing environment. Our framework for agile business integration shows the potential for SOA, but also the future research work that still has to be done. Even more so, Web services as today's SOA implementation, contribute to adaptive dependability to some extent, but also pose new challenges on typical end-to-end properties, like dependability *guarantees*. Therefore, future research is needed to develop SOC standards and technology to their full potential.

References

[1] Arjuna et al. Web services business activity framework, 2005. `http://specs.xmlsoap.org/ws/2004/10/wsba/`.

[2] A. Avižienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[3] S. Dustdar and M. Treiber. Integration of heterogeneous web service registries - the case of visr. *WWW Journal*, 2006.

[4] L. Froihofer, K. M. Goeschka, and J. Osrael. Middleware support for adaptive dependability. In *Middleware 2007—Proc. of the ACM/IFIP/USENIX 8th International Middleware Conference*, pages 308–327. Springer, 2007.

[5] D. Garlan and B. Schmerl. Model-based adaptation for self-healing systems. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 27–32, New York, NY, USA, 2002. ACM Press.

[6] IBM et al. Web services metadata exchange, 2004. `http://specs.xmlsoap.org/ws/2004/09/mex/WS-MetadataExchange.pdf`.

[7] IBM et al. Web services coordination, 2005. `http://www-128.ibm.com/developerworks/library/specification/ws-tx/`.

[8] J.-C. Laprie. Resilience for the scalability of dependability. In *Proc. 4th Int. Symposium on Network Computing and Applications*, pages 5–6. IEEE CS, 2005.

[9] S. M. Manson. Simplifying complexity: a review of complexity theory. *Geoforum*, 32(3):405–414, 2001.

[10] S. Murugesan and S. Dustdar. Web services-based e-business systems - a special issue. *International Journal of E-Business Research (IJEBR)*, 2(1), Jan – March 2006.

[11] J. Osrael, L. Froihofer, and K. M. Goeschka. What service replication middleware can learn from object replication middleware. In *Proc. of the 1st Workshop on Middleware for Service Oriented Computing at the ACM/IFIP/USENIX Middleware Conf. 2006*, pages 18–23. ACM Press, 2006.

[12] M. Shaw. "self-healing": softening precision to avoid brittleness: position paper for woss '02: workshop on self-healing systems. In *WOSS '02: Proceedings of the first workshop on Self-healing systems*, pages 111–114, New York, NY, USA, 2002. ACM Press.