The Development of Dependable and Survivable Grids*

Andrew Grimshaw[1], Marty Humphrey[1], John C. Knight[1], Anh Nguyen-Tuong[1], Jonathan Rowanhill[1], Glenn Wasson[1], and Jim Basney[2]

[1] Department of Computer Science, University of Virginia, Charlottesville, VA 22904, USA
[2] National Center for Supercomputing Applications (NCSA), University of Illinois at Urbana-Champaign, Champaign, IL, 61820, USA

Abstract. Grids should not just be facilitating advances in science and engineering; rather they should also be making an impact on our daily lives by enabling sophisticated applications such as new consumer services and support for homeland defense. This is not possible today because the poor grid dependability—which is tolerated by scientific users—would be unacceptable in critical infrastructure applications. This project aims at correcting this problem by developing technology that will allow grids to be used to provide services upon which society can depend. Through the Grid Dependability and Survivability Architecture (GDSA) and the Dependability Exchange and Specification Language (DESL), Grids will be engineered both to achieve high dependability and to permit assurance that high dependability has been achieved.

1 Introduction

Current grid systems provide services that are important to society, but our dependence will grow considerably over time reaching a point where they will become interwoven within the fabric of society. They will then have become critical infrastructure. Grid systems are dependent upon one another and will become more so. Grids providing financial services, for example, will depend on grids that support telecommunications, and grids providing health care services will depend on both financial services and telecommunications. The common need throughout all the various interdependent grid user communities is high levels of dependability; users must be able to depend upon the service they receive. This dependability will include reliability in some cases, availability in others, data integrity and confidentiality (security) in essentially all services, and possibly safety in some circumstances.

To engineer a grid to meet the expected high dependability requirements is likely to be technically infeasible or prohibitively expensive or both. Infeasibility derives from technical issues such as the reliance on commodity operating systems that are known to be vulnerable to cyber attacks. The expense derives from the very high cost of the replication of the basic components of the system, such as those providing

* This project is supported in part by the US National Science Foundation under grant SCI-0426972 (ITR for National Priorities, Network Centric Middleware Services).

V.S. Sunderam et al. (Eds.): ICCS 2005, LNCS 3515, pp. 729 – 737, 2005.

communications, storage and computing, that will be required if continuous availability is demanded.

The alternative approach endorsed here recognizes explicitly this fundamental trade-off between dependability and the resources and technology needed to provide it by making the next generation of grids *survivable*. Informally, a survivable system is one that provides one or more alternate services (different, less dependable, or degraded) in a given operating environment if the primary service cannot be provided because of attacks or failures. The key approach behind a survivable system is that each of these alternate services is designed to cope with potentially a different class of faults. By constructing or incorporating elements of the system (such as those implementing the primary functionality) with less provision for coping with faults than normally might be preferred, a survivable approach *may take the implementation of such a system from a complexity and cost level that is infeasible to one that is feasible*. The potential loss of service that ensues is dealt with by providing alternate services when the primary implementation is unavailable. Users receive a higher value from the system either in the form of reduced cost or increased functionality options.

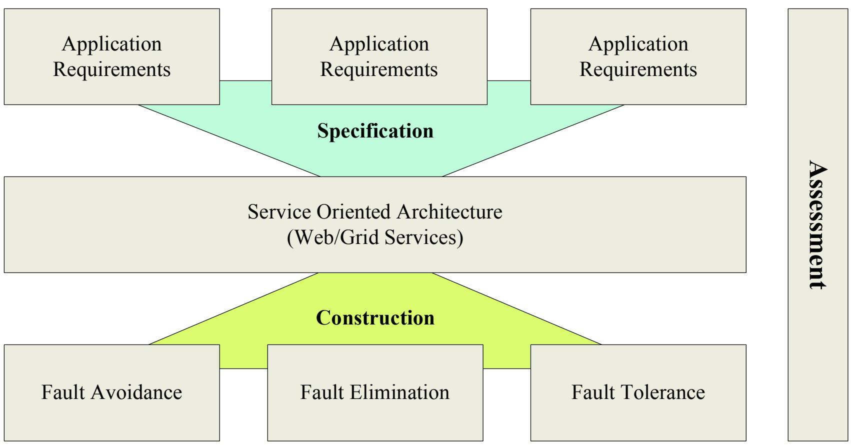

Fig. 1. Specification, construction, and assessment of dependable applications in service-oriented architectures

A survivable approach has the added advantage that it maps naturally to a development environment in which applications are built in part by incorporating existing grid elements from *different administrative domains*, an inherent characteristic of the grid paradigm. The specification of the alternate services advocated by a survivable approach provides an intellectual framework by which to analyze and incorporate existing grid elements with the inevitably different characteristics in multiple dimensions, e.g., different policies on the availability of the provided grid facilities, different expected average and peak load demands, different security policies, and different service availability guarantees. In such an environment, the dependability requirements for a specific application will be met in part by a process of resource discovery combined with the composition of services with the requisite levels of dependability. Note that oftentimes the straightforward composition of existing services will not be sufficient to meet the stated dependability requirements.

Our approach to the development of survivable grids is illustrated in Figure 1. At the bottom of the figure are the techniques commonly used to construct dependable systems: fault avoidance, fault elimination, and fault-tolerance. These techniques need to be enhanced to deal with the world-view that pervades anticipated grid systems and to implement the survivability concept. In the middle of the figure is the Service-Oriented Architecture (SOA). One of the hallmarks of an SOA is that of service composition: the construction of new services by composing, often dynamically, existing web services into new services. Available grid facilities change over time as circumstances change within administrative domains. Thus, for example, the hardware resources that can be provided by a specific administrative domain will depend on demands within the domain, failures that have occurred, and so on. We adapt the implementation of service provided to a given user at the time that the requirements are stated. At the top of Figure 1 are the applications, which have dependability requirements. Those requirements need to both be specified and tested against the dependability characteristics of the services the applications use.

This paper gives an overview of the key aspects of our approach. In Section 2, we describe the Dependability Exchange and Specification Language (DESL), an XML-based language used by grid elements to specify their dependability characteristics. In Section 3, we describe our dependability architecture and implementation, the Grid Dependability and Survivability Architecture (GDSA), based on a classic *data-driven* control system model. In GDSA a series of instruments will continuously monitor application and system behavior. Monitoring may be either direct polling, or subscribing to events of interest. The raw monitoring data will be processed, collected and distributed to appropriate dependability services, analyzed, and as needed both appropriate corrective actions may be taken and availability characteristics of components will be updated. GDSA and DESL build on our earlier work with Legion [1], Willow [2], OGSI.NET [3], and WSRF.NET [4][5] as well as the standards being developed in the Global Grid Forum. In particular, we will be constructing the prototype as a set of Web Services that are named with WS-References, and that fit within the context of the OGSA architecture. Section 4 contains an overall discussion on the engineering of dependable and survivable Grids. Section 5 concludes.

2 Grid Dependability and Survivability Architecture (GDSA)

GDSA is a data-driven control system architecture for dependability/survivability developed in the context of existing and evolving grid and Web Services standards, specifically the OGSA standards from the Global Grid Forum (OGSA-Program-Execution, OGSA-Core, OGSA-Security, OGSA-logging, etc.), and from the W3C and OASIS (WS-Security, WS-Context, WS-Notification, WS-Transaction, etc.).

To achieve dependability and survivability in a SOA given a dependability specification requires an architecture for detecting faults and responding to them. The problem can be thought of as a classic control system with sensors that monitor and provide status information to an analysis module that makes decisions on how to respond using an underlying control mechanism.

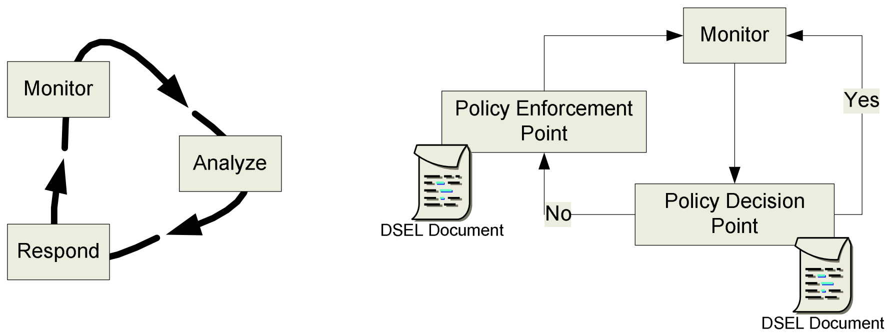

Fig. 2. Conceptual model for GDSA. The figure on the left shows the conceptual model as a classic data-driven control loop. The figure on the right shows more detail with the role of DESL as a specification language

The conceptual model is shown in Figure 2. Using various forms of monitors, the system detects errors, faults, state changes, or anomalous conditions. For example, message transmission fails because the receiver is not there, or a host fails, or an unauthorized user attempts access to critical data, or a monitor detects that the load on a host is above or below some threshold. Then analyze the events in the context of DESL specifications to see if the state with respect to the specification has changed. If conditions warrant, take some form of action to recover from the situation or exploit the change in state.

The PDP is a "policy decision point" [6]. We utilize this common terminology shared by many projects and approaches, for example XACML [7]. The PDP evaluates the current state based on information collected by the monitors or held in databases against policies and requirements carried in DESL documents. The PDP makes a "yes/no" decision. If the decision is "yes", no action is taken and the process continues. If the decision is "no" then the current state does not meet the specified requirements and action must be taken, i.e., a "policy enforcement point" (PEP) has been reached.

The PEP has as inputs the current state and a set of specifications, rules, trade-offs, and actions to take specified in a DESL document. The objective is to take an action to bring the state back to *some* correct state. We stress "some" because new correct state may not be what it was before, a degraded level of service may have been selected, or perhaps even no level of service at all.

The PDP and PEP combination can be thought of as a sort of application or service "manager" that is responsible for keeping an application or service delivering its specified QOS. Indeed below we will often call the combination an *application manager*. Keep in mind that the "application" may in fact be the enforcement of system level property, e.g., that no priority 2 jobs run anywhere if priority 1 jobs are waiting.

Second, there will likely be many application managers running concurrently in any real system. The application managers will have different, and often conflicting, objectives. Thus, like any large set of intertwined control loops some interference is to be expected, and unexpected behaviors may emerge. This is, we believe, a fundamental property of large scale, multi-organizational, grids. Different organizations will have different objectives. We are developing this technology from the existing prototype system that is in the Willow architecture.

The monitoring aspects of GDSA are provided by the classic publish/subscribe mechanism of WS-Notification and the OGSA-Grid-Monitoring-Architecture; polling techniques that acquire system meta-data and check component "liveness"; and by the use of ExoEvents [8]. ExoEvents are a mechanism for subscribing to a set of events that may occur in the call chain of a service invocation. For example, notify monitor_A if there is a "no such service" fault. The advantage to ExoEvents over classic publish/subscribe in a grid is that the set of services that may be used in executing a particular service may not be known *a priori*, making setting up the required subscriptions difficult. Further, the subscription is often needed only during the context of a particular call - not before and not after.

The "levers" (control mechanism) in GDSA consist of the additional porttypes defined services to support dependability and survivability techniques as well as the standard services provided by OGSA. Examples of additional porttypes are save-state, transfer-state, migrate, replicate, begin-epoch, end-epoch, etc. An example of a standard OGSA service call is a call on an OGSA-container to check the status of a service or instantiate a new service instance.

3 Dependability Exchange and Specification Language (DESL)

The essence of our approach is the specification and construction of a dependability framework in the context of grid SOA's. The first critical aspect is determining how dependability requirements and characteristics will be represented, inspected, and transformed, i.e. defining a language and a set of transforms on documents in that language. We define an XML-based language, DESL, that will be used for the specification and exchange of dependability and survivability characteristics, requirements and actions. We refer to services that process DESL documents as "DESL engines".

Although the exact structure and use of DESL is still emerging, we imagine two use cases for DESL documents. First, DESL may be used as a means to export a service or application's dependability and survivability characteristics. Clients may examine these documents to determine if they wish to engage a service. Service composition engines may use these documents to determine which services to connect in order to meet the overall dependability properties of an application. DESL may also be used to express the dependability and survivability requirements that a service requires of its hosting environment or its clients. For example, a DESL document may describe how its hosting environment should manage a group of replicants or when to rejuvenate a service [9]. In addition, DESL can describe required properties of a service's client, e.g. clients must be in the same survivability mode as the service to invoke methods on the service. In this use case, DESL can be seen as a more detailed version of WS-Policy [10].

We further refine the DESL language as used to export properties of a service. Consider the DESL document used to describe the grid service MyProxy (or a MyProxy-like service that might be available in the near term) [11].

```
<DESL xmlns:desl="http://gcg.cs.virginia.edu/DESL/02-21-04">
    <desl:Supports>
        <desl:SurvivabilityModes>
            <desl:Mode name="Normal">
                <desl:Operations>... normal delegation of credentials ... </>
            </desl:Mode>
            <desl:Mode name="NationalEmergency">
                <desl:Operations>
                    ... limited delegation creds to non-essential personnel
                </desl:Operations>
                <desl:Triggers> ... order from Dept. Homeland Security </>
                <desl:Effects>
                    ... desc. of "cost" of mode, e.g. reduced service for
some
                </desl:Effects>
            </desl:Mode>
        </desl:SurvivabilityModes>
        <desl:DependabilityInformation>
            <SoftwareProcess> ... </SoftwareProcess>
            <Availability> ... </Availability>
        </desl:DependabilityInformation>
        </desl:Supports>
    <desl:Requires>
            ... requirements of service: sub-services, performance, etc.
```

Fig. 3. A sample DESL document for MyProxy illustrating the basic document structure and the subcomponents

The DESL document in Figure 3 describes the supported "survivability modes" of the services as well as dependability properties of the service, both static (e.g. the software process used to build this service) and dynamic (e.g. the availability of the service, perhaps measured by the underlying container) in the <Supports> section. The <SurvivabilityModes> section of the document contains tags to describe the operations/actions that take place when the service is in that mode, the trigger event(s) that cause a service to get into that mode, and some description of the effects that users of the service can expect when the service is in that mode, i.e. what is the "cost" of being in a given mode. In the above example, when a national emergency is declared by the Department of Homeland Security, the MyProxy service will issue limited credentials to non-essential personnel (the effect being to prioritize grid operations from essential clients). The <Requires> section, shown for completeness, would contain any information needed by the MyProxy service from the fabric on which it depends (e.g. other services, or its underlying container) or from clients that access it.

Documents in DESL will be exposed, collected, transformed, and analyzed by "dependability services" such as the application managers above that are responsible both for making dependability assertions (based on the compositions of services that are used), for reading and interpreting assertions made by other services, and for making choices about trade-offs among multiple service implementation options.

4 Engineering Dependable and Survivable Grids

A critical obstacle to constructing dependable and survivable grids is that grid service developers are experts in neither dependable systems nor the protocols and messaging standards that run today's service-oriented grids. A supporting cast of tools, libraries and other grid services with which developers can turn their service logic into dependable and survivable grid services must be provided. To the extent possible, developers should only need to implement their application logic. However, the introduction of survivability concepts will require the mapping of survivability modes to grid services. In some cases, this mapping can be automatically performed by the container on behalf of the service, e.g., to enforce policies that limit external resource consumption or policies that can be enforced at the message level. In other cases, the developer will need to program the service to understand the various modes supported. In yet other cases, the mapping can be implemented by placing a proxy in front of existing services; this proxy would intercept a request to transition to mode X and map it to messages on the back end services.

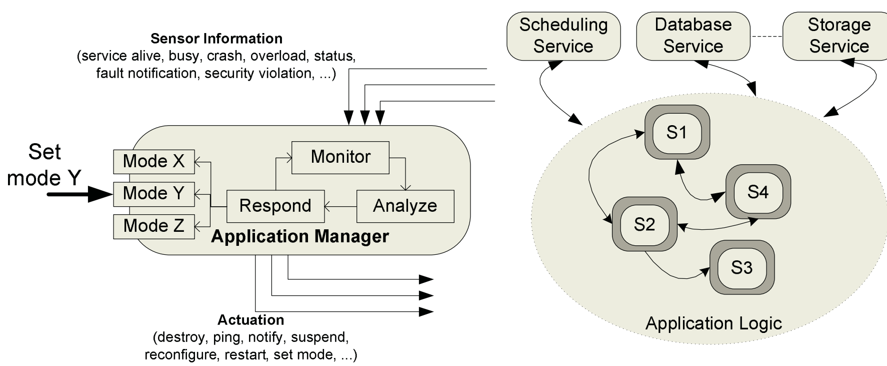

Fig. 4. Structure of a survivable grid application. Services S1-S4 represent the actual application logic. The other services provide support essential to the execution of the underlying application

Figure 4 illustrates a possible structure for a survivable grid application. The application logic is implemented by a set of communicating grid services labelled S1 through S4. The shaded region surrounding the services S1-S4 denotes that they are contained within a hosting environment, such as .net or J2EE. All other services represent the supporting cast needed to create and execute the application.

The application manager plays a vital role in the management of the application. It implements a DSEL engine and is responsible for: (a) receiving information from sensors, e.g., status information such as network and host load, network bandwidth prediction, liveness notification for S1-S4, fault notifications, security violations; (b) analyzing the stream of sensor information and taking appropriate actions, e.g., transitioning from one survivability mode to another; and then (c) issuing actuation commands, i.e., sending messages to either application services S1-S4 or other services. Note that transitions between survivability modes can be the result of external (and authorized) requests, themselves possibly issued as a result of another control loop, or as the result of an administrative decision to transition all applications within a grid domain to a given operating mode, e.g., "National Emergency".

5 Conclusion

In this paper, we have argued that a new approach is needed to construct Grids that serve as critical infrastructure. We have presented Dependability Exchange and Specification Language (DESL), used to express dependability properties and constraints, and on the overall architecture, the Grid Dependability and Survivability Architecture (GDSA). Through DESL and GDSA, Grids will be engineered both to achieve high dependability and to permit assurance that high dependability has been achieved.

References

[1] A.S. Grimshaw, A.J. Ferrari, F.C. Knabe and M.A. Humphrey, "Wide-Area Computing: Resource Sharing on a Large Scale," IEEE Computer, 32(5): 29-37, May 1999.

[2] J. C. Knight et. al., "The Willow Architecture: Comprehensive Survivability for Large-Scale Distributed Applications," Intrusion Tolerance Workshop, DSN-2002 The International Conference on Dependable Systems and Networks, June 2002.

[3] G. Wasson, N. Beekwilder, M. Morgan, and M. Humphrey. OGSI.NET: OGSI-compliance on the .NET Framework. In Proceedings of the 2004 IEEE International Symposium on Cluster Computing and the Grid. April 19-22, 2004. Chicago, Illinois.

[4] Humphrey, M., G. Wasson, M. Morgan, and N. Beekwilder (2004). An Early Evaluation of WSRF and WS-Notification via WSRF.NET. *2004 Grid Computing Workshop (associated with Supercomputing 2004).* Nov 8 2004, Pittsburgh, PA.

[5] G. Wasson and M. Humphrey. Exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments. 2005 International Parallel and Distributed Processing Symposium (IPDPS 2005), Denver CO, April 4-8, 2005.

[6] Yavatkar, R., Pendarakis, D. and Guerin, R. 2000. A Framework for Policy-based Admission Control. IETF RFC 2753. http://www.faqs.org/rfcs/rfc2753.html.

[7] Organization for the Advancement of Structured Information Standards (OASIS). Extensible Access Control Markup Language (XACML) Version 1.0. OASIS Standard, 18 February 2003. http://www.oasis-open.org/committees/xacml/

[8] Nguyen-Tuong, "Integrating Fault-Tolerance Techniques into Grid Applications," Department of Computer Science, Doctoral Dissertation, University of Virginia, August 2000.

[9] K. S. Trivedi, K. Vaidyanathan and K. Goseva-Popstojanova, “Modeling and Analysis of Software Aging and Rejuvenation, “ IEEE Annual Simulation Symposium, April 2000.
[10] D. Box, et. al. Web Services Policy Framework (WS-Policy). Version of 28 May 2003. http://www-106.ibm.com/developerworks/library/ws-polfram/
[11] J. Novotny, S. Tuecke, and V. Welch. An Online Credential Repository for the Grid: MyProxy. Proceedings of the Tenth International Symposium on High Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.