Middleware-Based Software Infrastructures for the Transparent Composition of System Properties

Priya Narasimhan
Institute of Software Research International
School of Computer Science
Carnegie-Mellon University, Pittsburgh, PA 15213-3891
priya@cs.cmu.edu

Abstract

As applications become more complicated, and yet demand even more system properties, such as reliability, real-time, security, $etc.$, software developers face the challenges of building real-world critical applications without unduly increasing the application complexity. This paper presents a novel middleware-based infrastructure that allows multiple system properties to be composed and provided simultaneously to the application. By operating transparently, the infrastructure allows unmodified middleware-based applications to reap the benefits of these system properties, without any increase in the application's complexity or any additional effort on the application programmer's part.

While processors are becoming faster, and memory is becoming cheaper, the computer applications of today and of the future are becoming more complicated. The challenge that software developers face, and will continue to face, is to build useful real-world software applications while avoiding the pitfalls that accompany any increase in the application's complexity. This becomes increasingly difficult as applications place more stringent demands – in terms of the *system properties* ("-ilities", such as reliability, scalability, interoperability, real-time, security, performance, $etc.$) that they require – that today's commercial off-the-shelf (COTS) infrastructures are simply not capable of satisfying.

Thus, if an application programmer today wants his/her application to exhibit one of these system properties, he/she cannot rely on the infrastructure to provide support for this system property and must, therefore, resort to some other means. For instance, consider a system property such as reliability (fault tolerance). With current practices, even for commercially available fault tolerance technologies, application programmers tend to mix the fault tolerance logic with the application logic in proprietary ways. While this might make their applications more reliable, there are many significant concerns from a software development and productivity perspective:

- The complexity of the application now increases significantly,
- The application programmer must now understand the intricacies of fault tolerance (in addition to, of course, understanding the application logic) in order to incorporate the necessary mechanisms into the application,
- There results an explosion of proprietary fault-tolerant solutions, quite often re-inventing the wheel to produce custom systems,
- The proprietary programming practices (both the home-grown and the commercially available solutions) are too expensive to be affordable, and mission-critical applications, such as 911 emergency systems that urgently require fault tolerance, will simply forgo reliability on the grounds of cost, and
- Reliability is really a system-level property, and requires consideration of the entire system, and cannot be provided by considering the application alone.

Thus, the resulting fault-tolerant application now becomes more expensive to build, more difficult to maintain, more difficult to extend, particularly because application programmers must now be trained, and become proficient, in areas outside their expertise, namely, fault tolerance and system-level issues.

Unless we develop an alternative methodology that addresses these problems, software developers ten years from now will continue to face these issues, and quite possibly for far more complex and for more demanding applications than those of today. Finding a solution to these problems becomes even more imperative when we realize that, every day, more new applications are being developed that will eventually turn into the legacy applications of the future, bringing with them all of this challenging "baggage". Thus, any alternative methodology that we propose in order to address these software development challenges must accommodate both current and future legacy applications.

One approach would be to delegate the responsibility for providing the system property to the infrastructure (rather than to the application), with the requirement that the infrastructure should be designed to support the system property *transparently* to the application. Furthermore, the infrastructure development would rest in the hands of infrastructure/system designers who are experts in the domain of that specific system property. The advantage of this approach is that application programmers would then be free to focus on the application logic (which is, after all, their domain of expertise), and will not require training in order to exploit the invisible infrastructure.

Another infrastructural design requirement stems from the fact that applications are increasingly called upon to exhibit more than one system property. For instance, it is no longer sufficient for an application to be equipped with reliability alone; it must also exhibit other desirable features, such as security, real-time, scalability, *etc.* This means that the infrastructure now must contain mechanisms for each of the individual system properties, and in addition, some means of combining them to produce useful services to the application. Unfortunately, composing diverse system properties is not a straightforward exercise – it involves an in-depth understanding of the individual system properties, an analysis of the trade-offs in combining them, and more importantly, a new, and more $holistic$, approach to analyzing, developing and evaluating the resulting system (consisting of both the infrastructure and the supported application).

Building such a composable dependable infrastructure is complicated by the fact that the infrastructure must be able to support applications that are developed using any one, or several, of the available commercial off-the-shelf tool-kits, architectures and programming models. In particular, there exists a wealth of applications based on middleware technologies such as Enterprise Java Beans (EJB), Common Object Request Broker Architecture (CORBA), Jini, JavaRMI, Distributed Component Object Model (DCOM), *etc.*. The number of such middleware choices increases daily, as variants of the basic middleware architectures are continuously being developed.

To avoid the proliferation of middleware-specific infrastructures that essentially accomplish the same objectives, I aim to develop a single composable communication infrastructure that underlies, unifies and provides desirable system properties transparently to the various middleware-based solutions. This is possible because (a) while each system property might be realized via different mechanisms for every middleware technology, the principles underlying that implementation of the system property are essentially very similar, and (b) there exist mechanisms that allow different middleware technologies to interoperate, and (c) there exist mechanisms that allow the transparent addition of system properties to each of these middleware technologies.

Thus, with applications clamoring for a higher out-of-the-box quality of service, the challenges in future software research will be centered around infrastructures that can

(i) Allow for the flexible composition of various system properties, such as reliability, security, real-time, $etc.$,

(ii) Allow for the customization of each system property (when considered individually, as well as when considered as a part of a composition) to suit specific application needs,

(iii) Hide the difficult system property-related issues from the application, allowing the application to enjoy a higher quality of service transparently,

(iv) Transcend, and remain independent of, the differences in architecture or programming model ($e.g.$, Java $vs.$ CORBA) in order to provide a single framework for composing various system properties, and

(v) Be evaluated and validated, in terms of useful metrics (that are applicable to system properties when considered individually, or as a part of a composition) that are widely applicable to various kinds of applications.

This paper presents the a novel transparent infrastructure that allows security, reliability and real-time to be composed in a customizable and flexible manner. I have elected to focus on three system properties – reliability, security and real-time – and to address the infrastructure-level issues in the context of applications based on middleware technologies such as CORBA, Java, EJB, DCOM and Jini. The ultimate goal of this research is to develop software infrastructures that can compose various system properties in an adaptive and custom manner, in order to support the production of more efficient, dependable software, without unduly increasing the application's complexity.

1 Composing System Properties

Developing a software infrastructure that exhibits even a single system property, such as reliability, is an involved and length process, requiring significant expertise in the area of that system property. What complicates this further is the fact that the software and applications of today and of the future will not be satisfied with just one system property, and are likely require multiple system properties simultaneously.

Furthermore, to stay competitive in the marketplace, software vendors need to differentiate themselves, and one common approach is for them to include more system properties into their offerings. The problem with this is that, in their hurry to place more attractive offerings on the market, vendors end up making compromises in the product development process that often results in products that do not support even a single system property correctly.

Currently, there does not exist a sufficient understanding of the *system-property space*, each of whose dimensions represents a specific system property, such as reliability, security, $etc.$. When different system properties are "married" to produce new compositions, the combined system-property space needs to be defined and understood more clearly.

For the sake of clarity, in the following text, I have chosen two system properties, namely, real-time and reliability, and addressed three specific aspects – classification, trade-offs and evaluation – in the context of the composition of these two system properties for a middleware-based infrastructure. This exercise needs to be undergone when any two system properties are composed, and when yet another system property (such as security) is added to the existing composition.

1.1 Classification and Applicability

Each system property has its own set of values; for instance, reliability can vary from weakly consistent replication (where availability of a server is more important, even if the data is stale) to strongly consistent replication (where the availability of the server and the integrity/consistency of the data are equally important). Thus, the design of the software infrastructure must factor in not only the composition of real-time and reliability, but also the fact that there are varying degrees of both real-time and reliable support.

There is a glaring lack of understanding and work in the classification of the composition of system properties that such next-generation infrastructures must address. Consider the composition of reliability, real-time and security for a CORBA-based infrastructure. The Fault-Tolerant CORBA specification [6], the Real-Time CORBA specification [7] and the Secure CORBA Service specification [8] have each attempted to classify their respective system properties, individually, by providing various levels of quality-of-service. However, none of these specifications addresses, and nor were they intended to address, how each individual property is impacted or perturbed by the addition of another system property.

The process of analyzing the resulting composition of real-time and reliability system properties involves the clarification of the two-dimensional system-property space, with one dimension being real-time (with various points on the real-time axis representing the degrees of real-time support) and the other dimension being reliability (with various points on the reliability axis representing the degree of reliable support). The design of the infrastructure, in the context of this real-time reliable system-property space, must take into account: (a) any well-understood design techniques that already exist in specific points in the system-property space, (b) the "useful" points in this system-property space that the infrastructure must support, and (c) the new properties that emerge in the system-property space due to the composition of various levels of real-time and reliability, respectively.

What emerges from such an analysis of the system-property space is a range of feasibility points for building real-time reliable infrastructures. A desirable side-effect of this analysis is the identification of target applications for each of these feasibility points. For instance, what kinds of applications can benefit from soft real-time and high reliability? This classification of the application space is vital to application developers because "one does not fit all" – the real-time reliable infrastructure that one would use to support a nuclear reactor is quite different in its requirements from the one used to support a network-based printer. Defining the target applications provides the badly-needed insight and guidance to infrastructure and software developers of the future, and allows them to pick the right infrastructure that suits the needs of specific application.

1.2 Trade-offs

When two different system properties are combined, there are likely to be trade-offs in the composition. For instance, a real-time system requires the the application's operations to be ordered so that deadlines are respected, while a reliable system requires the application's operations to be ordered so that data consistency is respected. To support both real-time and reliability, the infrastructure must necessarily reconcile the different ordering of operations in a manner that provides for both real-time reliable application behavior. However, such an ordering may not always be possible, and there might exist specific scenarios where either the reliability or the real-time system property is violated. In such scenarios, the

resulting, possibly degraded, quality-of-service needs to be well-defined so that the application knows what to expect from the infrastructure.

The Architecture Tradeoff Analysis Method (ATAM) [3] has looked at precisely these kinds of design trade-offs in developing architectures for three specific system properties, namely, modifiability, performance and availability. When analyzing the composition of real-time and reliability by extending the ATAM or other techniques, an in-depth understanding of each of the constituent system properties is essential. The trade-off analysis is complicated by the fact it involves (a) exhaustively listing all of the scenarios that might result from the composition of real-time and reliability, (b) assessing the risks and the degraded quality-of-service that might arise for either real-time or reliability, or both, in certain scenarios, and (c) providing solutions for scenarios where there appears to be violent conflict between the requirements for real-time and reliability. The infrastructure must then be developed to embody the results of the trade-off analysis, and to allow for new scenarios to be factored in, when they arise.

1.3 Evaluation and Validation

There still remains a lot of work to be done in the area of defining metrics for evaluating each individual system property. For instance, every fault-tolerant system developer defines his/her own notion of a metric for evaluating the reliability of his/her system; the problem is that there currently exists no single universally-accepted way to evaluate all reliable systems. There are currently efforts underway to define benchmarks for dependability, particularly by the IFIP WG 10.4 Dependability Benchmarking Special Interest Group [2]. However, defining such metrics for evaluating real-time and security, even as individual properties, is still in its primitive stages, particularly for middleware-based architectures.

This complicates the evaluation of the composition of system properties, such as real-time and reliability. Because there exist no well-defined metrics for evaluating even the individual system properties, there certainly do not exist any metrics for evaluating their composition! It is my hope that the development of middleware-based infrastructures for composing various system properties will remedy this deficiency, and will provide new insights into defining metrics and criteria for the evaluation and validation of real-time reliable secure systems.

2 Architectural Considerations

When developing such composable middleware-based infrastructures, there were three distinct aspects that I needed to consider from an architectural, rather than a system property, perspective: (i) transparency to the application, (ii) customization for various applications, and (iii) interoperability across various middleware technologies.

2.1 Transparency

Developing such software infrastructures such that their very presence, and their normal operation, are completely invisible to the application, makes for considerable savings in terms of development time, development cost, maintenance time, ease of maintenance, $etc.$ True transparency implies that the application should not need to be re-compiled, re-linked, or re-written, to take advantage of the infrastructure.

Unfortunately, infrastructure developers have to work much harder to build "black-box" infrastructures that are transparent to the application. However, there exist various ways of attaching infrastructures, at run-time, to existing applications, without requiring any modification of the application. One such approach is to use *software interceptors* [5], which are essentially user-space operating-system "hooks" that allow the dynamic modification of application behavior, and addition of capabilities to applications, at run-time. Software interceptors are a powerful mechanism, and have been used successfully in transparently adding reliability and survivability to CORBA applications [4], and in adding validation and security [1] to COTS architectures.

Using an interceptor, an entire sub-system can be attached transparently to an existing application, allowing the application to be enhanced without the application being aware of it. Because it exploits standard operating-system facilities, an interceptor can be implemented in a manner that is independent of the middleware that it underlies. Thus, we can attach a real-time, secure, reliable infrastructure to a CORBA/Java/EJB/DCOM middleware application in an identical manner. Another aspect of interceptors which is of considerable value in the composition of system properties, is that interceptors can be "chained", with each interceptor in the chain contributing towards the addition of one specific system property. Thus, not only do interceptors provide transparency, but they also foster the run-time composition of additional properties.

2.2 Application-Specific Customization

The reliability requirements for an embedded reliable system are different from those of an enterprise transactional system. Thus, because "one size does not fit all", the infrastructure must be tunable and customizable to fit the needs

of the specific target application. However, this is a contradiction in terms because exposing APIs for tuning the system properties of the infrastructure necessarily breaks the transparency that we are trying to promote for the infrastructure.

Transparency makes software development faster and more reliable, but does not allow the infrastructure to cater to a specific application's needs readily. Customization necessitates some interaction between the application programmer and the infrastructure in order to tailor the infrastructure appropriately; the danger, of course, is that, by giving application programmers (who are not necessarily experts in the infrastructure) more control over infrastructure-level issues, the resulting system may be more prone to errors. For instance, a fault-tolerant infrastructure needs to strike a balance between transparency and customization that allows application programmers to reliable software rapidly, while still permitting them to tailor the reliability to their application's needs.

One solution is to develop the infrastructure to operate at different levels of service, and to allow the user to configure the level of service that he/she desires, $e.g.$, a reliable infrastructure can present the user with a choice of active (state-machine) replication or passive (primary-backup) replication for the application objects. Once the user has chosen the desired configuration, the infrastructure assumes control once more, and continues to operate correctly. This is preferable because the user influences only the deployment-time configuration, but not the transparent run-time operation, of the infrastructure.

2.3 Transcending Different Middleware

One of the goals in designing the infrastructure was that it should interoperate seamlessly across diverse middleware-based technologies. Fortunately, the Internet Inter-ORB Protocol (IIOP), a TCP/IP-based protocol defined as a part of the CORBA standard, is widely supported by CORBA, EJB and JavaRMI middleware. There also exist DCOM-to-CORBA software bridges that allow the translation of IIOP into DCOM protocols.

By focussing on the IIOP-specific interface that is common to all of these these middleware technologies, and then using that interface for the interceptor to attach the infrastructure, a single composable infrastructure can be made to underlie CORBA, EJB and JavaRMI applications transparently. With some work, the infrastructure can penetrate to service DCOM applications as well.

3 Conclusion

Applications are becoming increasingly more complicated, and yet more needful of critical system properties, such as reliability, security, real-time, $etc.$ Providing these system properties without increasing application complexity is a challenge. This paper presents a novel infrastructure, whose facets include transparency (so that application programmers do not have to worry about implementing the system properties themselves), composability (so that more than one system property can be provided simultaneously), and interoperability (so that a single infrastructure can serve applications running over different middleware technologies). The key contributions of this infrastructure are the the understanding of the trade-offs involved in composing diverse system properties, as well as the transparent support for these system properties that leads to considerable savings in software development time.

References

[1] R. Balzer and N. Goldman. Mediating connectors. In *Proceedings of the 19th IEEE International Conference on Distributed Computing Systems Workshop*, pages 73–77, Austin, TX, May 1999.

[2] IFIP WG 10.4 Dependability Benchmarking Special Interest Group. *http://www-2.cs.cmu.edu/~koopman/ifip_wg_10_4_sig/*.

[3] R. Kazman, M. Klein, and P. Clements. ATAM: Method for architecture evaluation. Technical report, CMU/SEI-2000-TR-004, Software Engineering Institute, Carnegie-Mellon University, August 2000.

[4] P. Narasimhan. *Transparent Fault Tolerance for CORBA*. PhD thesis, Department of Electrical and Computer Engineering, University of California, Santa Barbara, December 1999.

[5] P. Narasimhan, L. E. Moser, and P. M. Melliar-Smith. Using interceptors to enhance CORBA. *IEEE Computer*, pages 62–68, July 1999.

[6] Object Management Group. Fault tolerant CORBA. OMG Technical Committee Document formal/2001-09-29, September 2001.

[7] Object Management Group. Real-time CORBA. OMG Technical Committee Document formal/2001-09-28, September 2001.

[8] Object Management Group. Security service specification version 1.7. OMG Technical Committee Document formal/2001-03-08, March 2001.