Holistic QoS – Transaction Systems Can Achieve Extreme Performance and Scalability While Enjoying Flexibility and Openness

A Submission to the Ninth International Workshop on
High Performance Transaction Systems (HPTS)

by Robert Vavra and Roland Angvall, Unisys Corporation

April 8, 2001

Abstract

Today's large transaction processing systems achieve extreme performance and scalability levels through careful ongoing optimization of their behavior. The implementation-hiding principle of component-based systems provides a great deal of flexibility to change systems over time, but seriously inhibits the optimization of systems. Non-functional requirements such as performance and scalability should be treated with similar attention and formality as functional requirements. This can lead to explicit Quality of Service (QoS) specifications for hardware and software components of a system, enabling a whole-system "holistic" view of QoS and its optimization.

Hand-tuned Systems Achieve Extreme Performance And Scalability

Today's large Transaction Processing (TP) systems are typically the result of many years of careful work tuning and optimizing the behavior of the systems. The optimization work seeks to achieve extreme performance and scalability levels from high-end hardware/software configurations. Unisys has deep experience in such optimization work, both in creating its own TP solutions and in assisting clients who create and run their TP systems on Unisys servers.

The people who perform such TP system optimization draw on a broad array of knowledge and skills. They routinely analyze application behavior and performance at system, macro, and micro levels. They look at resource usage patterns and follow processing paths across all software and hardware components of a system: applications, reuse libraries, external services, middleware, inter-process messaging, TP monitors, database managers, OSs, processors, memory, I/O channels, networking, disks, even storage farms.

At a micro level, optimization experts consider the efficiency of the highest volume transaction paths. They look at processor instruction counts and types, memory access patterns, data locking and contention, inline coding, I/O queuing, and more. They often study the source code of middleware and OS modules as well as application modules, looking for optimization opportunities. They might recommend special "fast-path" handling of the most commonly used functions, creating alternate code paths for optimized cases that are separate from the code paths for general cases.

At a macro level, optimization experts consider the efficiency of coarse-grained components and services. They look at the relative efficiencies of various services and usage patterns, advising application designers on how to make efficient use of available services. Sometimes they create software modules that implement and encapsulate desirable usage modes, so application designers can simply reuse them. (Unisys clients typically call these reusable modules "middleware" even though they are application-specific and bear little relation to the industry term "middleware.") Optimization experts also identify middleware and system services that are not needed by a specific TP system, and either disable or remove them to avoid unnecessary overhead.

At a system level, optimization experts create a view of the resource management algorithms across the entire system. They seek to understand current performance and scalability limits, and look for different configurations or usage patterns that would relieve such limits. They might see that algorithms in various components are working at cross-purposes to one another. In such cases they re-tune or replace the algorithms to work in concert. They might identify situations in middleware or system services where general-purpose algorithms are making pessimistic assumptions that aren't really necessary for a specific TP system. In such cases they introduce special-purpose algorithms that take advantage of design constraints enforced by other parts of the system.

Component-based Systems Hide Too Much

A key principle of component-based development is that users of an object, component, or service do not need to know (and should not know) its implementation details. Applying this principle can greatly improve the flexibility of TP systems to change over time. It facilitates the introduction of new and replacement components rapidly and safely. It also enables components to be acquired from multiple independent sources.

However, this principle is a serious inhibitor to optimization of TP systems. It denies optimization experts the ability to do the kinds of tuning and optimization work required to achieve extreme performance and scalability. At micro, macro, and system levels, optimization experts are not able to look at the information they need or make the changes necessary for efficiency improvements.

The implementation-hiding principle could also be seen as an inhibitor to creation of efficient components in the first place. It creates a false sense of freedom for component suppliers to change algorithms and performance characteristics at any time, as long as the component interfaces remain constant. When component suppliers do not know the usage patterns around their components and the common processing paths through their components, they do not know what to optimize.

The challenge at hand is to enable optimization both within and across components, without giving up the flexibility and openness introduced by implementation-hiding.

Non-functional Requirements Are First-class Citizens Too

Requirements specifications typically contain both functional requirements (what functions a system or component must perform) and non-functional requirements (characteristics or attributes that a system or component must exhibit while performing its functions). Both kinds of requirements need to be taken into account when creating and selecting components of a system.

To take a simple example, consider transmissions for use in motor-powered land vehicles. A successful transmission supplier needs to know not only the functional requirements (power, number of speeds, gear ratios) but also the non-functional requirements (duty cycle, ease of use, range of operating temperatures). It is unlikely that a single implementation of a transmission would meet the non-functional requirements of these vehicles, or that a vehicle designer could select an appropriate implementation without knowing its non-functional characteristics.


Here is a partial list of non-functional requirements drawn from an actual TP system architecture spec.

Non-functional Requirement	*Definition*
Availability	The degree to which a system or component is operational and accessible when required for use [IEEE 90 – Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990]. Availability requirements are portrayed in the form of three categories that correspond to the level of criticality for the processes: **Critical** The process has the highest possible criticality and requires near perfect system availability. **Essential** The process is extremely important but can tolerate brief periods during which the system is unavailable **Non-critical** The process is important, but is not time critical and it can be deferred, so it can tolerate periods during which the system is unavailable Whereas availability can be expressed in terms of both system and end-user availability, for the purposes of this system, availability must be viewed from an end-user point of view, unless otherwise specified. Criteria for restarting the business solution are defined as a function of the percentage of users affected by the loss of availability.
Flexibility	The ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed [IEEE 90]. The business solution is flexible to the extent that it readily accommodates changes while at the same time retaining a consistent performance profile. The initial flexibility of the business solution is a very significant factor in the ability to subsequently extend the capability of the business system.
Maintainability	The ease with which a software system or component can be modified to correct faults, improve performance, or other attributes, or adapt to a changed environment [IEEE 90]. The application is capable of being maintained in a predictable, consistent, and cost effective manner.
Performance	The application is capable of meeting stated performance goals, expressed in terms of system-local and end-user metrics. Performance requirements are portrayed in the form of three categories that correspond to the level of criticality for the processes: **Critical** The process has the highest possible criticality and requires near instantaneous responses to input. **Essential** The process is extremely important but can tolerate slightly longer delays in responses to input. **Non-critical** The process is important, but is not time critical and it can be deferred, so it can tolerate periods during which performance is degraded.
Scalability	The ease with which a system or component can be modified to fit an increasing workload or throughput requirement. The business solution is capable of supporting an ever-increasing workload with a consistent response time.

Many non-functional requirements flow directly from characteristics that are required of the business processes that a TP system supports. This has two key implications:

- A given function of a TP system might be under different non-functional requirements when invoked by business processes of different criticality. Therefore some indication of criticality might need to be present at run-time to govern resource allocation or to choose among alternative implementations.

  Example: An airline inventory/reservation system supports four classes of users (shown in priority order): in-house call center employees, who have a Service Level Agreement for 0.5 second response time; external travel agents, who have a Service Level Agreement for 2 second response time; frequent flyer program members who log in through the public web site; and the general public through the public web site. To meet the performance requirements under heavy load, the TP system must honor the priorities in all resource allocation decisions: selection of

messages from input queues, process and memory allocation, process and thread dispatching, database locking, in-memory data structure locking, I/O initiation, etc.

- A non-functional requirement might be strong enough to override some functional requirements, especially when a TP system is under heavy load or is running in a degraded (partial failure) configuration. Again this might require some indication of process criticality to be present at run-time so the functional logic can decide whether to perform full or partial processing.

  Example: An airline check-in system has a functional requirement to confirm seat allocations and issue boarding passes for all flight segments when a passenger checks in for the first segment. However, the check-in system also has a non-functional requirement to complete the check-in process for a single passenger with no checked luggage in 30 seconds or less (due to constraints on the number of agents and the amount of counter space at a given airport). If the check-in system cannot confirm seat allocations for continuation segments within the required time, perhaps because of network congestion, it should issue the first boarding pass only and let the passenger check in again at the next stop.

Some aspects of the non-functional requirements appear explicitly in the low-level design and code of a TP system, for example, process priorities and time limits. But many aspects of the non-functional requirements are implicit in the design and configuration of the overall hardware and software system. Examples include the selection of caching algorithms, the use of parallel network and I/O paths, the allocation of logical database structures to physical disk units, and the provision of alternate fast-path implementations.

For large long-lived TP systems, the development and maintenance teams spend a large portion of their time and energy tuning and optimizing the systems to meet these non-functional requirements. It is common for a critical TP system to have as many as six people dedicated to tuning and optimization of the system. The optimization work goes on continually, even at times when no feature development is being done for the system.

In a monolithic TP system where designers and optimization experts have a whole-system view of the implementation algorithms and their interplay, it is possible to manage and track the satisfaction of non-functional requirements despite their implicit nature. Many have done so successfully.

In a component-based TP system where designers and optimization experts try not to have a whole-system view of the implementation, it would not seem possible to satisfy a large set of implicit non-functional requirements. To successfully achieve extreme performance and scalability, it might be necessary to make the non-functional requirements more explicit.

Holistic QoS as a Basis for Optimization

QoS – Quality of Service – is a networking term for guarantees of non-functional aspects of communication sessions such as throughput, latency, and predictability. Typically a communication user and a communication service provider negotiate QoS levels at the beginning of a session. The communication service provider uses the agreed QoS levels to allocate resources and select algorithms appropriately.

We believe that QoS concepts can and should be broadened from networking to address non-functional characteristics of all hardware and software components of TP systems – “whole-system” or “holistic” QoS. Large system designers and optimization experts have been working with QoS concepts all along, because they are important system-wide non-functional characteristics. But there has been a lack of ways to formalize QoS concepts and make them explicit.

We suggest that component specifications should address both functional and non-functional capabilities of components. Functional capabilities are expressed in Interface Definitions and Design Contracts, and the state of the practice is quite good in these areas. Non-functional capabilities could be expressed in QoS Definitions and QoS Guarantees, and should be supported fully by analysis, modeling, and design tools.

Working in top-down application design mode, a component specifier could write a QoS Definition for the capabilities a component needs to provide. This would be part of the normal system design process during which functional and non-functional requirements are allocated to parts of the system. Component suppliers could use both kinds of requirements when making design and implementation decisions. The aggregation of component QoS Definitions would describe the QoS of the whole system (holistic QoS) and would show whether the system could meet its non-functional requirements.

Working in bottom-up application assembly mode, a component supplier could write a QoS Definition for the capabilities a component implementation actually provides. This would reflect the algorithm choices made and the sub-component QoS capabilities expected. An application assembler could select certain component implementations and exclude others based on their QoS Definitions. Again, the aggregation of component QoS Definitions would describe the QoS of the whole system (holistic QoS) and would show whether the system could meet its non-functional requirements.

Working in optimization mode, a special tuning team can look at the QoS specifications along with actual system performance measurements. They can identify problem areas, and then work with appropriate application and component designers to improve the efficiency of the whole system (holistic QoS).

We also suggest that some aspects of QoS should be represented at run-time, to assist in negotiation and provision of appropriate service levels across all layers of the software and hardware system. Perhaps this can be done in a declarative fashion, separate from functional code, similar to the ways that security management and transaction management are handled in current component-based environments. The run-time representation of QoS needs, and the multi-level resource allocation decisions necessary to meet QoS guarantees, become even more important as TP systems broaden to include external (web-based) services.

Conclusion and Proposal

System designers and optimization experts need to be able to work in a component-based environment, achieving extreme performance and scalability, without compromising the beneficial implementation-hiding aspects of components.

Component suppliers need to be able to work in a TP environment of extreme performance and scalability, describing non-functional characteristics and efficient usage patterns, without revealing implementation details of the components.

The industry can take a big step in this direction by making QoS definitions an explicit part of component specifications, creating formal models of required and provided non-functional capabilities. This step will enable use of holistic QoS concepts at design- and run-time to optimize efficiency across an entire TP system.