



12/06/2004 1:19 PM

“ Fads companies and products come and go, and in this sea of ideas and products it is often difficult to decide if it is worthwhile to invest the time to investigate a new product. With software tools, all too often the complexities of the tool obscure the problem that the tool tries to address. In this respect QP [Quantum Platform] is different to most. The fundamental concepts of QP are not new. Miro's contribution and what the QP offers is valuable because it integrates the basic concepts of Hierarchical State Machines (HSM's), a minimal real time framework and a tiny preemptive multitasking kernel, and packages them in a consistent, well documented and portable fashion. Most important however is accessibility. Users can get the book on Friday and begin to realize the benefits using industry standard C and C++ on Monday. The quality of the source code and its consistent use of design by contract (DBC) are also part of a coding methodology that confers benefits that may not be immediately apparent. Finally, by placing the code in the open source domain, developers and management can feel confident that the fundamentals are solid and will be further tempered and extended over time in the furnace of public scrutiny.



But all the hype aside, what is QP really, *and what does it offer me?*

As a software team lead, I want my team to be as productive as possible. Much of the challenge is to get the software team on the same page philosophically and to employ a consistent set of design and coding idioms. One





also needs to feel confident that the fundamentals are sound and can extend naturally as complexity grows in the problem domain.

As our company has matured in the use of "QP methodology", our software discussions and terminology now focus on actors, assignment of responsibilities, message content and design patterns. This has been a huge step forward. We have achieved a common language and graphical representations that help us to efficiently capture, comprehend and convey the most important aspects of a software design.

A little appreciated and easy-to-miss associated benefit is the use of design by contract (DBC). From this alone, the nature of bugs has changed. The whole difficult class of concurrency related bugs are largely absent. Typically, both conceptual and implementation bugs are detected earlier and fixed more quickly.

But don't be fooled — there is no "Silver Bullet". Complex problems are still complex and modeling remains non-trivial. Brooks writes of essential complexity — complexity that is inherent to the problem domain, and accidental complexity — complexity, which is introduced by the process, the model, and the toolset [Frederick P. Brooks, Jr., "No Silver Bullet: Essence and Accidents of Software Engineering", *Computer Magazine*; April 1987].

QP and the "QP methodology" have proven to be of great help in allowing the designer to focus and quickly refine models to address the essential complexity of the problem domain while minimizing many elements of accidental complexity.

QP is a great contribution to the software community and there are many things I like about it. It is perhaps also fair to comment on some aspects that we have been found through experience to be problematic.

To list these briefly:

Scalability: Basically the constraint on the number of actors, and the number of actors that can subscribe to an event. We changed the code to remove these restrictions at relatively minor cost to memory footprint.

Publish subscribe model: In several circumstances, it is much better to use directed publishing. Consider the enormous amount of traffic and impact on message queue sizes that results when changing QDPP to 63 philoso-





phers. A lookup table can be used to retain most benefits of loose coupling if this is critical.

Assignment of fixed actor priorities: This is at times more tricky than might be imagined.

Patterns of usage: We are still learning about usage patterns. What works and what does not. For example, subscribing and unsubscribing as entry and exit actions can lead to surprising order of execution.

Elements of HSM implementation: The current implementation is in some respects non UML compliant. For example, the implementation does not allow initial transition to a highly nested state and to subsequently occupy a less nested state.

In summary, these observations do not detract from the overall goals of the QP and I highly encourage developers to take the time for detailed study.

NOTE: Early versions of what is today QP, were developed and used by IntegriNautics in its core GPS receiver technology. A GPS receiver is an ideal testbed for the concepts of the QP, having significant domain complexity and requiring hard real time determinism, small footprint and low power. Subsequently the QP has been adopted across the company and is used in all products on a variety of OS platforms.”

— **Dr. Paul Y. Montgomery, Director of Engineering, Novariant Corp. (formerly IntegriNautics Corp.), Menlo Park, California**

