state that people think they can build solutions for tomorrow (next five years) are also live in an un-real world thus also being irresponsible.

We cannot impose today's paradigm to the builders of the future solutions because paradigms do change and tools supporting the paradigms change with them. Using today's language to implement yesterdays paradigm is not the right way to manage development lifecycle. Our best option is to state exactly what we are building today, a clear interface specification of our system and components separated from the actual implementation and algorithms. This is our best chance that tomorrow's software builders will understand what we have built and try to integrate our components into their solutions.

One of the major battle fields for technology war is the standards process. So far, there are two styles of standardization, the totalitarian and democratic. No matter what the style, two elements must exist to have a success: the moral authority (people trust that the technically best is produced); and commercial authority (that the commercialization of the standards can be enforced). So far most standards processes cannot achieve even one of these. As a consequence, the processes that all set out to reduce the technology wars intensify them.

How many times we come across an RFI or RFP which state that the solution must conform to a large list of open standards. Can those who wrote the RFI/RFP elaborate convincingly that there is a real business need to conform to these standards and they really believe that their need can be met, or it is just part of being good *corporate citizen*.

Many modern standards processes are flawed in at least two areas (apart from the lack of moral authority and commercial authority). First, we try to standardize things before a full free market competition is well underway and people's creativity and innovation is exhausted; second, we set a goal of plug and play (and we further mislead the users that the plug and play can be achieved free) or full integration. Plug and play can only be achieved when a thing runs out of its commercial value and becomes a true commodity. Whenever someone has the goal of producing the best X, they need to break out of the current paradigm. This often cannot be done in a plug and play manner.

So how to avoid war in the standards world? Are open standards useful? Yes, but as an architect responsible for delivering business solutions, the conformance to open standards should never be used as the criteria for success. To build a solution that minimizes the impact of war is our success criteria. In achieving this goal, we may choose to conform to those open standards that fit the problem.

What do we mean by conform? The purpose of conformance is to make sure that the components of solutions can interact. Our experience is that to harmonize the interaction of two components is more important than to conform to a standard. To achieve harmonization, we use the terms called maximum specification and minimum specification.

When we build a component, we should always consider that we are building a specialization of a much large solution. One way to achieve this is to fully specify its interface and behavior. When others use our component they must understand exactly our design intent and the performance of our component—that is, that is, we always anticipate that someone will generalize our component for other purposes. Without this specification they will redevelop the component.

When we integrate with another component, we must treat the component we are going to integrate with as a generic component. An example is the interaction of buying a car; a car producer will always provide a full specification of the car (all the way down to the size of the bolts), while a consumer always provides a minimum specification when buying a car (such as color and number of seats).

This is particularly important when we integrate with a component designed with an older paradigm. The minimum specification makes sure that we don't try to implement that old paradigm using the new language. Instead, we must integrate the component by generalizing its behavior using the new paradigm.

To sum up, a responsible software developer should always try to avoid the technology war. The best way to do this is to manage the development lifecycle within a much larger environment, and to always anticipate the change of the technology and products.

An irresponsible software developer always try to pick winning horses, in most cases, before the race has started.

Let technologists fight the war, we need these wars so that their innovation and creativity energy will be fully displayed. This leads to better technology and products. But those build solutions don't care if the technology and products of today are gone tomorrow!

WAR AND PEACE —HOW TO AVOID TECHNOLOGY WARS?

GRAHAM CHEN

Source of Publication—**This paper was presented as a Panel Presentation at the 6th IEEE/IFIP Network Operations and Management Symposium, New Orleans, LA, USA in February 1998. It is to appear in the Thresholds column of the Journal of Network and Systems Management, Plenum Press.**

Technology wars arise when multiple technologies and products exist and the migration from one to another yields a competitive advantage. This usually happens when there is a discontinuous innovation that changes the way we do business. Technology wars are caused by rough and tumble free market forces. They often lead to healthy competition and better products. Sometimes all they lead to are delays in solving problems. Technology wars can be good, but only when we manage their impact.

Let's consider an issue even bigger than a technology war, namely the lifecycle management of software development for the telecommunications industry. Since this industry has been at the leading edge of the adoption of software technology, its experiences are often carried directly into other industries. To manage the software development lifecycle one must manage the lifecycles of its paradigms and products.

A paradigm is a way of solving problems. It is a belief system that the software activities should be conducted in a particular way. The typical examples of paradigms include batch processing, mainframe architecture, relational database, artificial intelligence and more recently and more relevant to this discussion the distributed object computing. Paradigms typically have a much longer lifecycle. They typically have three stages:

A paradigm almost always spends it infancy in universities and research institutes. During this period, there are lots of wars but largely irrelevant to the industry. Because it is far too early to change any of industry's software development behavior before this stage is complete. Many paradigms die before reaching to this point.

When a paradigm reaches to a mature age, people start to build products, start to build software solutions. This is where war starts. Some technology or products will emerge as the winner of the war and some will not. This is where standard processes start which lead to more wars. When we reach the apex of the maturity, we start to see the *dummy's guide to XYZ* in the bookstores.

In time paradigms die. Batch processing, once a most powerful computing paradigm has died gloriously. Today's paradigm is tomorrow's legacy. There is nothing wrong with this. We just have to manage this process.

Paradigms are realized in products and their releases. At any stage of the lifecycle of a paradigm, there are different products with various levels of customer acceptance.

To avoid a technology war, we need first recognize that we are building a business solution that will outlive individual technologies and products by a large margin. We need then to consider how to minimize the negative impact of the technologies and product on the longevity of our solutions. Further, we need consider whether we can spend all our effort to pick the best technology and then adapt our solutions to it. The challenge is to find the technology that will naturally evolve in step with the needs of the business.

A paradigm may be flexible enough to encapsulate some features and concepts of an older paradigm, a product is never meant to be able to bridge two paradigms.

When people build today's business solutions, they must be responsible for today's operation and for yesterday's systems. This holds for even those people building for new telecommunications companies born from global deregulation. Even they do have a green field opportunity, they must fit into the existing worldwide telecommunications infrastructure. Today is tomorrow's yesterday. There is no shortage of examples that some of new telephone companies having to face the past with the same level of difficulty as the long lived ones.

Although our major responsibility of building software is for today and yesterday, it is obvious that people who don't consider tomorrow in their solutions are living an irresponsible life. However, I would go further to