As the complexity of embedded systems evolves, real-time Java has come to play a more central role in large-scale applications that demand higher levels of abstraction, portability and dynamic behavior. Such applications are taking on roles in management of network infrastructure, automation of manufacturing processes and control of power generating equipment. To meet these demands, real-time Java vendors have moved increasingly into the mission-critical domain.

The accelerating move into mission critical and the expected eventual integration into safety-critical applications have increased the need to assure that Java can deliver reliable operations without exceeding resource constraints. Mission-critical computing involves a broad spectrum of applications. Resource-limited applications such as deep-space probes, remote planetary exploration and communications satellites stand in contrast to the resource-rich applications of telephone switches, semiconductor manufacturing and air-traffic control.   Given the breadth of domains spanned by mission-critical software, one challenge is to establish a sufficiently large area of common ground on which to base meaningful standards for mission-critical computing.

Part of what makes mission-critical software difficult is the need to integrate large numbers of independently developed components, each satisfying different mission objectives. Over time, most mission-critical systems scale upward in response to evolving requirements and expanding capacity. Each step of the integration process must preserve the key characteristics of each component so that no mission-critical objectives go ignored.