INTRODUCTION

Aim of this paper is providing a personal vision on some key prerequisites for current software design, namely dependability and adaptability, and introducing some ideas we are currently developing at the University of Antwerp and the Institute for Broadband Technology. The key message we want to convey is that, today, conceiving software design without the adaptability concern is simply not possible anymore, as this course of action would not match the systems and environments our software is meant for.

The Computer Era may well be thought of as a series of “revolutions”: The 19th-century concept of computer was that of a mechanical engine, initially intended to compute, quickly and reliably, tables of polynomials. That which now
Adaptation and Dependability and Their Key Role in Modern Software Engineering

sounds like a trivial achievement, in times past was but a dream whose fulfillment had puzzled mankind nearly since the beginning of its history. This achievement was to be reached only in 1855 through the design of Babbage and the craft of the Scheutzes. And this marked indeed an actual revolution, that of mechanical computing, which may be well summarized by Babbage’s famous quote “I wish calculations had been executed by steam”: For the first time in human history, a tool to perform calculations more quickly and reliably than a human being had been realized.

The 20th Century witnessed a variety of those revolutions, soon to provide a new meaning to the word “computer”. One such revolution was brought about by the advent of vacuum tubes, in the Forties. The supercomputer of those times was the ENIAC, with a weight of about 30 tons and an availability quite disappointing when considered out of the historical context the ENIAC had made its appearance in. Clearly there was room for improvements and several further “revolutions” were to be expected.

Further revolutions sprang from new concepts, now concisely summarized by terms such as “compiler”, “virtual machine”, “object orientation” or “service orientation”. Each of these concepts brought about a sheer revolution, for it modified the meaning and the use we made of computers thereafter.

Each of these revolutionary steps marks a fundamental leap in the history of computing and of the influence of computers in human society. Each step allowed new services to be conceived while, in turn, these services called for additional requirements and adjustments of our “view” to the concept of “computing.”

As a consequence of this, each of those steps also marks the need for new models, both for the computer and for its system software.

We are currently in the middle of another important step in this progression of revolutions, namely the one marked by the spread of personal computing facilities that allow their services moving with us and our goods: It is the so-called “wireless revolution”. In this paper we investigate on some of the key requirements for software components meant for wireless or other dynamically changing environments, and describe the main ideas of a prototypic system that we are currently designing as a tool to support the development and execution of mobile services.

The structure of this paper is as follows: In the second section we introduce our target problem. Adaptive systems are discussed in the third section. The fundamental services to be provided by any architecture for adaptability are conjectured in the fourth section. Two examples are given in the following one. A concise survey on adaptability is given in the sixth section, followed by our conclusions in the last section.

SERVICES AND PROGRAMS

In the following we consider a service as a set of manifestations of external events that, if compliant to what agreed upon in a formal specification, can be considered by a watcher as being “correct”. Moreover we refer to a program as a physical entity, stored as voltage values in a set of memory cells, which is supposed to drive the production of a service. Goal of software engineering is being able to set up of a robust homomorphism between a service’s high-level specification and a low-level computer design (the program).

More formally, we say that for some functions \( f \) and \( g \),

\[
\text{Service} = f \ (\text{program}), \ \text{program} = g \ (\text{specification}),
\]

\[
\text{Service} = g \cdot f \ (\text{specification}).
\]

Building robust versions of \( f \) and \( g \) is commonly a very difficult job.
Related Content

The Knowledge Society Applications: The RRR Language Machines
[www.igi-global.com/chapter/knowledge-society-applications/28319?camid=4v1a](www.igi-global.com/chapter/knowledge-society-applications/28319?camid=4v1a)

From Europe to China: Adapting Courseware Generation to a Different Educational Context
[www.igi-global.com/chapter/europe-china-adapting-courseware-generation/56072?camid=4v1a](www.igi-global.com/chapter/europe-china-adapting-courseware-generation/56072?camid=4v1a)

Modularity and Complex Adaptive Systems
[www.igi-global.com/chapter/modularity-complex-adaptive-systems/24184?camid=4v1a](www.igi-global.com/chapter/modularity-complex-adaptive-systems/24184?camid=4v1a)

Emergent Behavior in Complex Organizations: Teamwork on Software Development Process
[www.igi-global.com/article/emergent-behavior-in-complex-organizations/136301?camid=4v1a](www.igi-global.com/article/emergent-behavior-in-complex-organizations/136301?camid=4v1a)