Adaptation as a new requirement for software engineering

Reference:
De Florio Vincenzo, Blondia Christian.- *Adaptation as a new requirement for software engineering*
Proceedings of the First IEEE WoWMoM Workshop on Adaptive and DependAble Mission- and bUsiness-critical mobile Systems (ADAMUS 2007), Helsinki, Finland, 2007 - s.l., 2007
Handle: http://hdl.handle.net/10067/662190151162165141

Adaptation as a New Requirement for Software Engineering

Vincenzo De Florio, Chris Blondia
University of Antwerp
Department of Mathematics and Computer Science
Performance Analysis of Telecommunication Systems group
Middelheimlaan 1, 2020 Antwerp, Belgium
Interdisciplinary institute for BroadBand Technology
Gaston Crommenlaan 8, 9050 Ghent-Ledeberg, Belgium

Abstract

The so-called wireless revolution is producing new services based on the concept of mobile computing. We remark how mobile computing technologies call for effective software engineering techniques to design, develop and maintain mobile services*, i.e., services that are prepared to continue the distribution of a fixed, agreed-upon quality of service despite of the changes in the location of the client software and the characteristics of the environment. Hence novel paradigms are required for software engineering so as to provide effective system structures for mobile services while keeping the design complexity under control. In this paper we discuss the problem and propose one such structure.*

1. Introduction

Aim of this paper is providing a personal vision on some key prerequisites for any mobile computing services design, namely dependability and adaptability, and introducing some ideas we are currently developing at the University of Antwerp.

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 sounds like a trivial achievement, in times past was but a dream whose fulfilment had puzzled mankind nearly since the beginning of its history. This achievement came true 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[1], 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[2]. Clearly there was room for improvements and several further "revolutions" were to be expected[3].

[1]To fully appreciate the extent of this revolution, let us reprint here an excerpt from an article by Brisse [2] celebrating the 1855 Paris Universal Exhibition, where the Scheutzes' machine was shown to the public for the first time: "This machine solves equations of 4th and even greater degree; operates in any numerical system [. . .] The scientists, boasting their computation capabilities as a miracle of natural law, will be soon taken over by a simple machine that, under the nearly blind guidance of a common man and by means of custom movement, is going to dig the infinite outer space with a security and depth way greater than that of scientists. Any man able to formulate a problem and having at his disposal Mr. Scheutz's machine will have no need for Archimedes, Newtons, or Laplaces [. . .] This *quasi-intelligent machine* not only computes in a few seconds what normally would require hours; it also prints the obtained results, adding the advantages of a neat calligraphy to those of computations *with no chance for errors*". See also [4, 1].

[2]This excerpt from a report on the ENIAC activity [14] gives an idea of how dependable computers were in 1947: "power line fluctuations and power failures made continuous operation directly off transformer mains an impossibility [. . .] down times were long; error-free running periods were short [. . .]". After many considerable improvements, still "trouble-free operating time remained at about 100 hours a week during the last 6 years of the ENIAC's use", i.e., an availability of about 60%!

[3]See for instance the following statement from *Popular Mechanics*, 1949: "In the future computers will weigh at most 1.5 tons"!

- 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 brough about a sheer revolution, as it modified the meaning and the use we make of computers.

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 to sit on top of a mobile system 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 the paper is as follows: In Sect. 2 we introduce our target problem. Adaptive mobile systems are discussed in Sect. 3. The fundamental services to be provided by any architecture for adaptability are conjectured in Sect. 4. An example is given in Sect. 5 while Sect. 6 concludes this text.

2. 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, that 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 well known as being a though job.

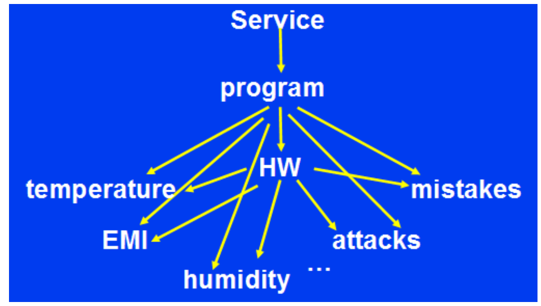

Figure 1. An expansion of the dependence relation.

We now concentrate on the range of g (the software set) and for any two systems a and b, if a relies on b to provide its service, we say

$$a \rightarrow b.$$

We call this relation as the “dependence” between two systems. Clearly it is true that e.g. Service $\rightarrow$ program, program $\rightarrow$ CPU, and program $\rightarrow$ memory. Figure 1 provides a possible expansion of the dependence relation.

As evident from that picture, dependences call for *dependability*, i.e., a fundamental property to achieve *dependable services*, which has been defined by Laprie as “the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers” [9]. A dependable service is then one that persists even when, for instance, its corresponding program experiences faults—to some agreed upon extent. From the above definitions we propose to consider F-dependable services (resp. F-dependable programs, systems, etc.) as those that persist despite the occurrence of faults as described in F, F being a set called the fault model.

What is F exactly? It is a set of events whose occurrence we consider as possible and likely to hinder the quality of the service distribution. An important property of F is that it is a model of an environment E where the service (or better, its corresponding program) is operating. Clearly an F-dependable service may tolerate faults in E' and may not those in E'': an airborne service may well experience different events than, e.g., one meant in a primary substation [13].

Obviously the choice of F is an important aspect to successfully achieving service dependability. Imagine for instance what may happen if our fault model F matches the wrong environment, or if the target environment changes its characteristics (e.g. a rising of temperature due to a firing). But the key point we remark herein is—what if the service *moves*? In all these cases, lacking provisions to accomodate for the changes in the environment and in the corresponding fault model, *a failure may occur*, i.e., an interruption of the service may take place. We summarize the above reasoning with this famous quote by James Horning:

> "What is the most often overlooked risk in software engineering? That the environment will do something the designer never anticipated." [8]

In other words, if in the early days of modern computing it was to some extent acceptable to have lengthy service disruptions due to the occasional faults, being the main task of computers basically that of a fast solver of numerical problems, the criticality associated with many tasks nowadays appointed to computers demands high values for properties such as availability and data integrity.

At the same time and for similar reasons nowadays provisions are required to accomodate for the changes occurring "around" a service, i.e., in all the components that the service is dependent on. We call this property *adaptability*.

3. Adaptive Mobile Systems

As a consequence of the advent of mobile computing, a system's environment has become a variable, which translates in a strong need for *adaptability*. In what follows we provide our personal vision to adaptive services and the current state in our quest for an effective solution to this problem.

Ideally, we would require our services to be structured as "X-dependable services", where $X = f(E)$ can change dynamically when e.g., the service is moved to another environment or the environment mutates. X should be indeed considered as an $X(t)$, that is, as a dynamic system. We consider also as an important prerequisite for an effective crafting of those services that the expression of adaptability and dependability concerns should not increase complexity "too much," so as to avoid possible "bottlenecks of system development" [10]. In other words, whatever solution we may come up with, it must keep complexity bounded and under control.

Our proposal is then to consider an adaptive system as a triple

$$AP = (\mathbf{F}, \mathbf{FT}, \mathbf{E}),$$

where $\mathbf{F}$ expresses the functional concerns (the service), $\mathbf{FT}$ is some fault tolerance provision to withstand the faults in a fault model $F_{\mathbf{FT}}$ and $\mathbf{E}$ is a set of environments (to be described later on).

Let us suppose that program $(\mathbf{F}, \mathbf{FT})$ distributes a certain $F_{\mathbf{FT}}$-dependable service and that a family of fault tolerance strategies, $(\mathbf{FT}(k))_{k \in K}$ be available. Furthermore, let us assume that program $(\mathbf{F}, \mathbf{FT}(j))$ distributes a $F_{\mathbf{FT}(j)}$-dependable service.

Then we can translate the problem of crafting an adaptive system into that of designing an architecture that *senses* the environment and, each time the environment changes, *changes* program $(F, FT(j))$ into a program $(F, FT(k))$. If the resulting $F_{\mathbf{FT}(k)}$-dependable service matches the new environmental condition, and this is true for a set of environments $\mathbf{E}$, then we have realized an adaptable service.

In the following section we briefly sketch the main components of an architecture for adaptable services.

4. Components of an architecture for adaptability

Our conjecture is that any effective architecture for adaptable applications should be structured on top of two basic services, namely *change detection* and *change reaction*. Our approach is summarized in Figure 2. We assume to have detection components, ranging from simple sensors providing raw data like temperature or heartbeats rate to scenario detectors able to correlate raw data and provide higher level, more structured information about the state of the subjects or properties being monitored. As soon as a relevant change is detected, we publish the event in a reflective shared space and check whether the change brings in a new scenario (e.g., a patient has fallen and is in need). The next phase is executing the actions attached to the detected scenario: this is done in our prototypic architecture by the Reactive component, a virtual machine interpreting a simple scripting language called Ariel [3]. Such an architecture could be used in the framework of projects such as "ARFLEX" (Adaptive Robots for Flexible Manufacturing Systems, IST-NMP 2-016680, http://www.arflexproject.eu) and IBBT project "End-to-end Quality of Experience" (https://projects.ibbt.be/qoe).

In practice, we envision the availability of a middleware component (MW) to update dynamically the $(\mathbf{FT}(k))_{k \in K}$ programs (we call them "recovery codes"), aided in this by a set of "change detectors" monitoring e.g., available energy, network load, or local or overall CPU usage. Figure 3 shows one such system.

A prototype of a compliant architecture has been described in [5]. Another approach, situated in the application-layer, reflects the environmental changes into the memory cells associated to so called "reflective variables" and achieves reactivity by assigning so called "refractive variables" [7]. This approach is being implemented these days in the framework of the above mentioned projects.

In the following we propose two examples to clarify how our vision of adaptive systems may provide a solution to seemingly contradicting quality of service requirements.

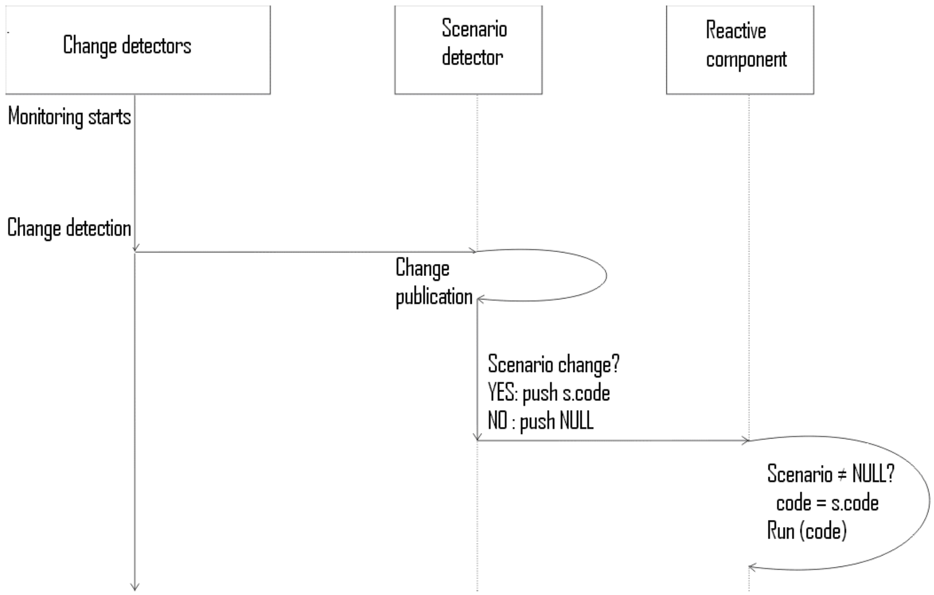

Figure 2. Main architectural components of a system to support adaptive services.

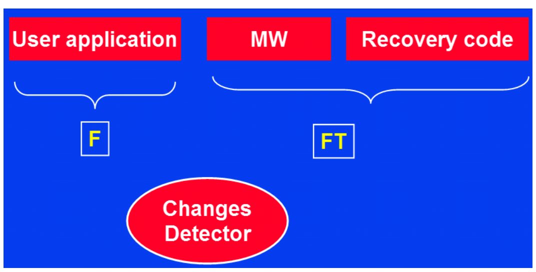

Figure 3. Main architectural components of a system to support adaptive services. "F" stands for functional component, while "FT" are the fault tolerance components in the architecture.

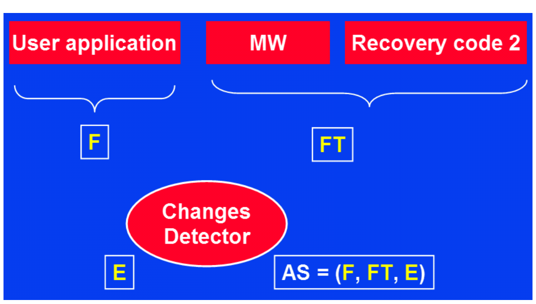

Figure 4. The change detector sensing a change in the environment E signals the middleware MW to push a new recovery code to match the new environment.

5. Examples of Adaptive Mobile Services: Adapting Voting Sensors and Adaptively-redundant Data Structures

5.1. Adapting Voting Sensors

Systems such as Body Area Networks (BANs) of wireless sensors are becoming more and more important for many reasons ranging from health care effectiveness to social security costs. In such systems patients are constantly monitored by mobile units that continuously transfer and publish the value of a set of vital parameters between a patient's location and the clinic or the doctor in charge [11]. As remarked in [5], a true effective service like this is based on the contemporary fulfillment of two seemingly contradicting requirements:

R1: (Hard) guarantees are required so that, whenever the patient is in need, the system is to trigger a system alarm (e.g., dispatching medical care to the patient).

R2: (Soft) guarantees are required, such that no false alarm is triggered when the patient is not in real need—the latter to reduce the service costs.

Clearly **R1** triggers the system alarm whenever any one of the sensors alerts or disconnects while **R2** does so only when the condition is confirmed by the occurrence of several sensor alerts. As explained in [5], a possible solution may be trading off between **R1** and **R2** through m-out-of-n majority voting, and trigger the system alarm after m out of the n sensor alerts. Whatever the choice of m, this approach is too unflexible because the choice of m is fixed ahead of the run-time. An alternative solution would be to set up a series of strategies, $(\mathbf{FT}(k))_{k \in K}$, each of which may consider a different environment (for instance "heartbeat = 70, temperature = 38°C, arterial pressure = 120") and compute an $m(k)$-out-of-n majority voting. The consequence of this strategy would be that we would decompose the space of events into a set of blocks with known characteristics (we call them "scenarios" in the cited paper) and provide the best strategy matching each of these blocks, basically decomposing an unstable environment like our sensor networks into a set of quasi-stable environments.

5.2. Adaptively-redundant Data Structures

Another important requirement of computing services is that of enhancing data integrity: Protecting data against transient and permanent memory faults. The current practice for data integrity provisions [12] is to take the fault model as a static choice, which means that our data integrity provisions (DIP) will have a fixed range of admissible events to address and tolerate. This translates into two risks:

1. overshooting, i.e., over-dimensioning the DIP with respect to the actual threat being experienced, and

2. undershooting, namely underestimating the threat in view of an economy of resources.

Adaptively-redundant data structures constitute an effective strategy to avoid these two risks. In such system the amount of replicas of our data structures changes dynamically with respect to the observed disturbances. We assume that a monitoring tool is available to assess the probability of memory corruptions of the current environment (this corresponds to the *change detection* service described in Sect. 4. *Change reaction* is in this case tuning the employed redundancy after the observed disturbances. We have built a prototype for such a system, which is described in full detail in [6].

6. Conclusions

We have introduced the problem of adaptability as a fundamental requirement for a system where the environment and hence the fault model varies over time. A model focussing on the anticipation of changes in the environment has been proposed. We have shown with two examples the potential of adaptive services as means to achieve a dynamic trade-off between seemingly contradicting requirements.

References

[1] Calculating by machinery. *The Manufacturer and Builder*, 2(8):225–227, August 1870.

[2] Léon, baron de Brisse. *Album de l'Exposition universelle de Paris en 1855*. 1875.

[3] Vincenzo De Florio. *A Fault-Tolerance Linguistic Structure for Distributed Applications*. PhD thesis, Dept. of Electrical Engineering, University of Leuven, October 2000. ISBN 90-5682-266-7.

[4] Vincenzo De Florio. *Course notes of Advanced Computer Architectures, part I*. Master programme in Artificial Intelligence. Katholieke Universiteit Leuven, 2002.

[5] Vincenzo De Florio and Chris Blondia. A system structure for adaptive mobile applications. In *Proceedings of the Sixth IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM 2005)*, pages 270–275, Taormina - Giardini Naxos, Italy, June 2005.

[6] Vincenzo De Florio and Chris Blondia. Adaptive data integrity through dynamically redundant data structures. Submitted to the 26th International Conference on Computer Safety, Reliability and Security (SAFECOMP 2007), Nuremberg, Germany, September 2007.

[7] Vincenzo De Florio and Chris Blondia. Reflective and refractive variables: A model for effective and maintainable adaptive-and-dependable software. Submitted to the 33rd EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA 2007), Lübeck, Germany, August 2007.

[8] James J. Horning. ACM Fellow Profile — James Jay (Jim) Horning. *ACM Software Engineering Notes*, 23(4), July 1998.

[9] Jean-Claude Laprie. Dependable computing and fault tolerance: Concepts and terminology. In *Proc. of the 15th Int. Symposium on Fault-Tolerant Computing (FTCS-15)*, pages 2–11, Ann Arbor, Mich., June 1985. IEEE Comp. Soc. Press.

[10] Michael R. Lyu. Reliability-oriented software engineering: Design, testing and evaluation techniques. *IEE Proceedings – Software*, 145(6):191–197, December 1998. Special Issue on Dependable Computing Systems.

[11] Philip E. Ross. Managing care through the air. *IEEE Spectrum int.'l edition*, 41(12):14–19, 2004.

[12] David J. Taylor, David E. Morgan, and James P. Black. Redundancy in data structures: Improving software fault tolerance. *IEEE Trans. on Software Engineering*, 6(6):585–594, November 1980.

[13] Unipede. Automation and control apparatus for generating stations and substations – electromagnetic compatibility – immunity requirements. Technical Report UNIPEDE Norm (SPEC) 13, UNIPEDE, January 1995.

[14] Martin H. Weik. The ENIAC story. *ORDNANCE — The Journal of the American Ordnance Association*, January-February 1961. Available at URL http://ftp.arl.mil/~mike/comphist/eniac-story.html.