Software Development Support for Ambient Assisted Living

Max Mühlhäuser

Telekooperation, FB Informatik, Technische Universität Darmstadt, Hochschulstr. 10
64289 Darmstadt, Germany
max@informatik.tu-darmstadt.de

Abstract. Key issues in software development support for Ambient Intelligence and Ubiquitous Computing are briefly discussed; special requirements in the context of Ambient Assisted Living are discussed.

Keywords: Software Development, Ambient Intelligence, Ubiquitous Computing, Ambient Assisted Living.

Development Support for Smart Environments and AAL

In the area of *ambient intelligence* (AmI) (which we consider synonymous to *ubiquitous computing*), complex and integrated distributed applications must be developed. Such complex applications are often called *smart environments.* Not surprisingly, software development for smart environments faces particular needs. The following five key needs have been identified at Telecooperation and met by our research:

(1) Harmonization programming paradigms: Smart environments are highly distributed (contain many components) and open (grow over time thru addition and exchange of components). A design-time distinction into clients and servers is usually inappropriate. These characteristics call for the use of a 'push' based communication model (driven by the occurrence of events rather than by consumers asking for data and synchronously communicating with the providers that marked the 'pull' model). The publish/subscribe programming and communication paradigm has become wide spread for the realization of this 'push' model. However, experience from our research and development tells that quite often in the software development 'daily business', events trigger a communication relationship between peers that remains stable and intensive for a period of time; programmers tend to leverage off the 'pull' model and stateful communication 'links' in these cases, resorting to 'socket programming' (TCP and the like) or 'remote method calls' (e.g., based on Java-RMI).

To make things worse, smart environments make use of continuous-media streaming quite often (voice and video transmission, etc.). The related communication paradigm and programming means are again different from the ones mentioned (push, pull). This means that without proper consideration, software developers are faced with the need to apply three different distributed-programming paradigms and means. Given the inherent complexity that smart environments already impose by them-

Dagstuhl Seminar Proceedings 07462
Assisted Living Systems - Models, Architectures and Engineering Approaches
http://drops.dagstuhl.de/opus/volltexte/2008/1469

selves, projects run a high risk of failing due to an unmanageable code base. Therefore, we find it absolutely essential to harmonize and integrate the three paradigms mentioned, offering to the software developer a single concept, interface, and set of programming means for all three paradigms.

(2) Harmonization of Resource Span: AmI marks the departure from the 'equal-resources' assumption that was fair to make in the past: although PCs and servers were often somewhat different with respect to resources, operating systems (with, e.g., virtual memory) and cheap hardware made software developers ignore resource differences; middleware developers hardly considered such constraints eithers – with well known effects on the size of frameworks like .NET™, Corba™, J2EE™, and so on. With AmI, several reasons make resource constraints an important concern, particularly the number or 'density' of components deployed (cf. sensor networks) demanding extremely low-cost solutions, energy consumption (cf. battery-powered unattended devices) and size constraints (cf. mobile phones). From a software development perspective, it is obvious that application developers should be supported as much as possible by the middleware and development tools in reflecting these constraints; the middleware should take over as much as possible of this task – and most notably, the middleware itself should require minimal footprint on low-resource devices while offering maximal support in the overall system. In our research, we provide several corresponding concepts, particularly the 'protocol heap' approach for auto-customizing the code segments installed on small nodes and a 'proxy' approach for transparently offloading functionality from small nodes to 'the net'.

(3) Exploitation of trusted, personal devices: with AmI, more and more daily tasks are assisted by computers. But if computers are involved in 'almost every step of daily live', then trustworthiness (in terms of reliability, privacy-protection, safety and security) becomes a key issue. Trustworthiness has a very important, if not dominant 'non functional' component. For instance, people *establish* trust as they use things over time and do not get 'disappointed'. We consider it absolutely crucial in this respect that users are endowed with a personal, ubiquitously carried device that they can call 'their own' and that no one else can intrude, control, or change. The more they establish trust with such a device, the more they become willing to hand over responsibility to it, especially with respect to representing them and acting in their stead in smart environments. Since we increasingly *depend* on computers for conveniently executing daily tasks, such a personal device must become our ubiquitous companion – which also means that it has to be as 'small' as possible. In our research, we developed and evaluated an elaborate concept for such a device; this concept was called 'Minimal Entity' (ME); the ME concept was designed such as to potentially interact with user-associable further devices (US) such as digital cameras or laptops, with smart items in the environment (IT), in adhoc wireless opportunistic networks (WE) and with the functionality provided by services in the background (THEY). The default minimal (bootstrapping) interaction modality proposed is voice, and apart from obvious ingredients like right-sized CPU, memory, and wireless communication, a ME is proposed to host cruicial data and functions necessary for computing the users context (location, head orientation, mode, …); thereby, the trusted ME controls the user's privacy, too.

(4) Additional Development Tools: The overall software development lifecycle for smart environments and their components is not much different from the one for any

other large software project. A few additional development tools are desperately needed, however. Among the tools developed at Telecooperion, the following are particularly worth mentioning: appropriate *distributed debugging and testing tools* (adjusted to the harmonized three programming paradigms); *visual development and testing aids, both for 2D* (cf. interactive floor plans that show the position and interworking of smart components) *and 3D* (cf. laboratory automation where the 3D positions and movements of smart components are important); *scripting and simulation aids* for quickly prototyping new smart components and for integrating them seamlessly with the already-deployed smart environment; *additional services* of general use, deployable with both the development and runtime environments (most important, a *context server* that harmonizes, e.g., different location/positioning services and pre-computes them for sophisticated inclusion in context-aware application components, but also a range of further novel services like an *indoor navigation service* etc.); *multimodal interaction* development aids that help to shield the core functionality of components from the large (and a priori unpredictable) range of I/O devices and modalities used at runtime (we developed, e.g., model driven architectures and suitable meta-models for separating general and modality specific functionality, hinting based approaches for combining flexible programmer-transparent interface generation with the need for programmers to influence user interfaces for critical issues, aesthetic concerns etc.).

(5) Intelligent Composition: current service description, discovery, and composition / orchestration approaches are not considered sufficient for combining the envisioned degree of ‚intelligence' and 'self organization' of future smart environments with the envisioned ease-of-use of means for composing and controlling the ensemble that makes up a smart environment. This area is still under heavy investigation at Telecooperation. With respect to intelligent self-organizing composition, we developed intelligent planning algorithms and are working, improved service description and scoping, and self-organizing service orchestration; with respect to user-friendly composition, contex-aware smart-workflow description and management aids and visual aids were developed.

In reflecting the above on AAL environments, the author emphasizes an *increased* importance of the five requirements cited, rather than *additional* concerns. A few examples shall be selected to support this claim: (i) the unavoidable intrusion of intimate areas of personal live makes trusted devices (cf. ME) and 'self determination of information flow' even more crucial; (ii) highly user-adapted, multimodal interaction is an even more deterministic issue; (iii) openness and 'gradually growing sophistication' must be reflected since many 'players' i.e. organizations, professions, etc. must be potentially involved and heavy cost constraints in the public health system call for organic growth as the need for assistance increases (or functions can be privately afforded); (iv) since patients may depend on services provided by the smart environment in a life-critical way, the reliability must be (among others) ensured by sophisticated and smart environment specific development aids. The following reference may be consulted as an entry point for further information.

Aitenbichler, E., Kangasharju, J. Mühlhäuser, M.: *MundoCore: A Light-weight Infrastructure for Pervasive Computing.* In: Pervasive and Mobile Computing. 332-361, Elsevier (2007).