Commentary on 'Software architectures and mobility: A roadmap'

Journal Item

Commentary on 'Software Architectures and Mobility: a Roadmap'

Michel Wermelinger and **Arosha Bandara**, Computing Department, The Open University, UK

Medvidovic and Edwards provide in their paper (henceforth referred to as the Roadmap) a good survey of the variety of architectural concerns and approaches to mobility. This commentary is intended to complement the Roadmap, discussing additional or alternative concepts, issues and approaches.

The ubiquity of mobile computing devices has enabled communication and access to information and services at an unprecedented scale. Novel application domains and technological advances make the boundaries between mobile, ubiquitous, pervasive and autonomic systems fuzzy. To meet user demands, applications are likely to include facets of several types of systems. We therefore start this commentary with a discussion on what mobility means (Section 1) followed by when does mobility impact architecture (Section 2). Finally, we touch upon two crucial aspects of mobile systems that can benefit from a principled architectural approach: autonomy (Section 3) and privacy (Section 4).

1 Mobility

We think that the distinction made in the Roadmap between mobile software and mobile systems is neither the most clear, because both definitions mention mobile software and hardware, nor the most helpful, because it does not guide the pursuit of the essential concepts that software architecture has to deal with. The distinction between logical and physical mobility strikes us as being more useful. While many systems will exhibit both kinds of mobility, each one poses separate challenges and requires different approaches, thus helping frame the discussion.

Physical mobility is just about the movement of hardware. A portable audio player is an example of a physically mobile system. Such devices must be portable, which leads to constrained resources: battery power, small screens, less memory than desktop computers, etc. However, being mobile doesn't mean the same as being location-aware: an audio player can be moved around, but it doesn't know it is being moved and hence doesn't make any use of its current location. The only relevant 'location' for such devices is whether they are connected to a computer from which they receive new files and software updates. Physically mobile devices that can communicate while on the move necessarily are location-aware, even if only in a very indirect way through sensing changes in the available wireless connections.

On the other hand, logical mobility is about the movement of code across (possibly fixed) computational hosts. This sets a completely different kind of challenges from physical mobility. Among them are the security and privacy risks of allowing external code to execute on our hardware: computer viruses are a striking example. Such non-functional concerns can have a great impact and Carzaniga et al (reference 9 in the Roadmap) argue that security has been one major factor why mobile agents have not been widely adopted by industry.

Besides the kind of mobility, the boundaries of the mobile system also have to be explicitly considered. The mobile system might be a single device (e.g. a smartphone), a variable set of interacting mobile devices (e.g. an ad-hoc network of laptops), or even a large system with a substantial non-mobile part managing the mobile components (e.g. the cellular telephony network). The scope of the system to be designed will obviously impact the architectural choices.

2 Architectural Impact

While both physical and logical mobility require new concepts and techniques, it is clear that mobile software (in the sense of logical mobility) is a more radical departure from the status quo than 'stationary' software in mobile hardware (in the sense of physical mobility). As the Roadmap shows, several models have been proposed for logical mobility, but calling them architectural styles only confuses matters, in our opinion. The

models only clarify the various logical mobility patterns that are possible, they do not provide any new styles, i.e. constrained configurations of architectural components and connectors. Vice-versa, nothing prevents a traditional style to be used for logical mobility: for example, a pipe and filter architecture might use mobile code techniques to allow filters to deal with unforeseen data. In other words logical mobility paradigms and architectural styles are separate conceptual entities, with a many-to-many mapping between them.

The two kinds of mobility and their requirements lead to different kinds of solutions, not necessarily involving software architecture. Solutions at the level of the programming language or library API can address part of the challenges of physical and logical mobility. For example, time-outs in remote procedure calls help cope with unreliable communication, class loading and object serialization help achieve weak logical mobility. At a higher level of abstraction, the solution might involve not any new linguistic constructs or libraries, but a systematic use of the available ones, akin to design patterns. For example, while the CHAM's basic constructs (rewriting of hierarchical multisets of terms) allow for dynamism of behaviour and structure to be expressed (reference 14 in the Roadmap), a more systematic use of certain patterns of terms and rewriting rules makes it easier to specify particular kinds of dynamic architectures [4].

Architectures needed to handle mobility can be sometimes taken from a non-mobile context. For example, architectures of embedded non-mobile systems might be partially adopted for physically mobile (and hence resource-constrained) systems, and the BlueStar architecture (Figure 13 in the Roadmap) is, apart from the device control product component that interacts with the mobile devices, a generic device management architecture that could be used as well to manage the configurations of networked PCs.

We thus feel that the intersection of software architecture and mobility, i.e. the set of *specific* mobility problems that require a *tailored* architectural solution, is smaller than the Roadmap authors consider. In some cases, a non-architectural solution is possible, in others the architectural solution is addressing problems that are not unique to mobile systems. Nevertheless, the crossroads of architecture and mobility include enough challenges.

We posit that, at the fundamental level, mobility challenges four core concepts of software architecture: connectors, style, drift and decay. Mobility leads to an ever-changing context for each device, and an overall network configuration in constant flux. In such a setting, where each device's and the overall system's architectures are fluid in order to adapt and interact with each other and the environment, novel and more meaningful notions of architectural drift and decay are needed. Without a fixed goalpost, how can drift and decay be detected, measured and acted upon? Likewise, the traditional notion of style as a pre-determined set of constraints on the allowed configurations has to be widened and become less rigid. The concept of connector also has to be extended, not only to coordinate transient interactions between components, but to encompass the coordination of the components' mobility. As the Roadmap mentions, FarGo provides an elegant solution to capture commonly occurring movement coordination patterns. However, CommUnity, also mentioned in the Roadmap, provides a far more expressive approach. By allowing the architect to specify the data type that captures the required physical and/or logical mobility space, and by allowing connectors to manipulate such locations, CommUnity allows connectors to coordinate both the behaviour and the movement of the interconnected components and make each one (behaviour and movement) dependent on the other, as it occurs in real-life mobile systems [5]. The challenge is to make the transition from such formal approaches to practice. As the bullet list in section 3.2.1 of the Roadmap clearly shows, even 'traditional' connectors are still far from being first-class citizens in implemented systems.

Moving from architectural concepts to actual architectures, the Roadmap highlights several domains where new approaches have been put forward to handle logical mobility (e.g. for software deployment) or physical mobility (e.g. through middleware). We will further expand on two aspects of mobile systems already mentioned in the Roadmap: autonomy and privacy. The reason for focussing on these is that although they also apply to certain non-mobile systems, future 'killer' mobile aplications will have to exhibit sophisticated forms of autonomy and privacy, or else they will fail to meet users' expectations. We therefore think these two aspects will provide for more exciting developments in the future than, for example, logical mobility.

3 Autonomic systems

Autonomic (or self-managing) systems have been studied over the past decade as a paradigm for managing the increasing complexity of software systems. The goal of autonomic computing is to automate the micro-

management tasks related to maintaining a running software system such that users can concentrate on higher level tasks such as analysing and specifying the macro-level management of the system.

Bradbury et al [1] surveyed the support for self-management in 14 approaches (based on graphs, process algebras, logic or otherwise) to dynamic software architectures. The survey is guided by a change process model, based on [2], comprising four steps that may occur in various orders: initiation of the change, the selection of the necessary changes, the implementation of such changes, and the assessment of the effect of the changes. The survey then examines how the approaches deal with these steps, e.g. if they allow for internal initiation (the defining criterion for self-management) and what kinds of constructs (like iteration and choice) can be employed in the implementation of changes. They find that none of the approaches allows unconstrained selection of changes (i.e. in most approaches the space of potential changes in reply to triggers is more or less constrained) and that most do not allow for distributed management of the changes.

We feel that it is in these two aspects that specific architectures for self-management can bring the best support, whereas other aspects (like specifying the actual change operations to carry out or accessing sensors to achieve context-awareness) are best supported by specific language constructs or library APIs. And indeed, several proposals go in the direction pointed out by Bradbury et al. On the one hand, approaches like ArchWare [3] and others allow for each subsystem to have its own change choreographer and hence allow for distributed change management. On the other hand, approaches like the RAS style and the Three Layer model for self-management mentioned in the Roadmap allow for more unconstrained planning of how to reconfigure the system in reply to changes in the environment. Whilst frameworks for engineering context-aware applications, such as the Context Toolkit [6] described in the Roadmap, are important contributions to this field, they do not provide the adaptation capabilities that are an essential part of a ubiquitous computing system.

More expressive and robust software architectures for autonomic systems include the Self Managed Cell (SMC) architecture [7] and IBM's Policy Middleware for Autonomic Computing (PMAC) [8]. The basic design principle of all of these architectures to provide a closed-loop feedback control mechanism for an autonomic system to monitor its state, perform some planning and analysis to determine if any adaptation is required and execute any adaptation actions. For example, the SMC architecture consists of a set of components that form an autonomous entity that automatically adapts to changes in the user's context and also component failures and sensor errors in a mobile system. The architecture supports composition and aggregation of multiple SMCs which means the different autonomous entities can dynamically come together into a coalition to achieve a common goal. The approaches for combining SMCs can be classified into different types of patterns – structural patterns (composition, aggregation and peer-to-peer), communications patterns (publish-subscribe, shared blackboard, diffusion, store and forward) and management patterns (hierarchical control, collaborative control, auction based control). An SMC could be instantiated to handle any of the goal, plan or component level adaptation described in the Three Layer model for self-management mentioned in the Roadmap, and therefore is considered to be a generic architectural pattern for self-management.

4 Privacy by Design

Mobile applications aim to provide unique experiences that are adapted to the particular context of the user. However, systems that realise this vision often overlook the impact on the user's privacy. Often there are no mechanisms for the users to control and manage their privacy within such mobile applications, forcing the users to constrain use of the mobile system and in some cases, leading to privacy violations. The Roadmap highlights the importance of privacy and security in mobile systems in terms of structural guarantees of trustworthiness such that users can be confident that sensitive data is protected. However, providing such guarantees is only possible if we first understand users' expectations of privacy and trustworthiness. Important questions in this context include: Do users actually understand how much of their personal information is being shared with others? How can software architectures ensure that privacy is an integral part of the design of mobile systems? Together with colleagues at Imperial College London, we are currently investigating these research questions as part of the EPSRC PRiMMA project, which aims to develop techniques for protecting the private information typically generated from ubiquitous computing applications from malicious or accidental misuse [9].

As part of this work we have been investigating the use of social networking tools via mobile devices. To better understand the threats to user privacy, and the social mechanisms that have developed around the use of social networking services to protect user privacy, we conducted a number of user studies [10]. Based on the feedback

from these studies, PRiMMA researchers are developing a suite of tools for users to control their data (in terms of the collection, storage and dissemination) while promoting awareness of privacy issues. This is based on a different approach, called Personal Containers, in which the current arrangement is inverted - with users, rather than third-party services, assuming responsibility and control over gathering, storing and disclosing their own data [11]. A Personal Container is a user-controlled server that gathers, securely stores and manages a person's digital data. Access to the stored data is facilitated through a variety of protocols such as POP3, IMAP, HTTP and XMPP. All accesses are controlled by the owner's privacy policy, enabling privacy-aware sharing of the stored data (Figure 1).

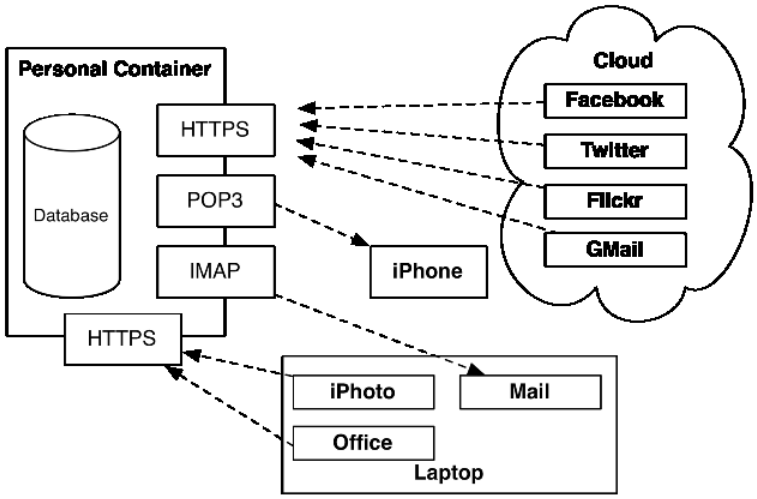

Figure 1: Personal Container Architecture, from [11]

The fundamental principles of the Personal Container architecture is to provide users with ownership and control over their personal data, together with mechanisms for accountability so that users are aware of how their data is being used by external entities. These principles are based on the concepts of notice, choice and consent, anonymity/pseudonymity, security, and access and recourse described by Langheinrich as essential requirements for privacy by design in ubiquitous computing [12]. The same principles underpin the Personal Data Vault (PDV) architecture being developed at the Center for Embedded Network Sensing at the University of California [13]. Whilst the Roadmap paper does not make any explicit reference to architectural solutions to the challenges of privacy, we believe that approaches that are based on the principles of privacy by design, such as the Personal Container and Personal Data Vault, demonstrate significant progress in developing architectures that can provide the necessary structural guarantees of privacy and security for both software designers and end users.

5 Concluding remarks

This commentary aimed to complement the Roadmap by focussing on two other classifications of mobility (physical vs. logical) and narrowing the scope of the impact of mobility on software architecture. We argued that only architectural approaches that 1) address the specific challenges of mobility (namely the change of physical or logical location and the ensuing change of context) and 2) add value beyond what linguistic and low-level design mechanisms can achieve, will make their mark and achieve effective progress. We suggested two research paths, each addressing those two points. One is to extend fundamental concepts (like connectors and architectural style) in order to provide the adequate conceptual tools to design and analyse mobile computing systems operating in a highly dynamic environment. The other is to concentrate on characteristics (like autonomy and privacy) that are essential for mobile systems to interact with unanticipated contexts. The key word is 'interact': mobile systems need not just be aware of the context, they also must set the rules by which they want the context to be aware of them. Hence the rising importance of privacy in an increasingly mobile world.

The Roadmap and this commentary just provide a *few* ways forward. The future of software architecture, mobility and their mutual influence will certainly prove to be far more interesting and diversified than any of us can predict.

References

1. J. Bradbury, J. Cordy, J. Dingel, and M. Wermelinger. A Survey of Self-Management in Dynamic Software Architecture Specifications. Proc. 1st ACM SIGSOFT Workshop on Self-managed Systems, pp 28-33. ACM, 2004.
2. J. Andersson. Issues in dynamic software architectures. In Proc. of the 4th Int. Software Architecture Workshop (ISAW-4), pages 111–114, 2000.
3. Cîmpan S., Leymonerie F., Oquendo F., "Handling Dynamic Behaviour in Software Architectures", In: Proceedings of European Workshop on Software Architectures, pp. 77-93, LNCS 3527. Springer, 2005.
4. M. Wermelinger. Towards a chemical model for software architecture reconfiguration. IEE Proceedings - Software, 145(5):130–136, 1998.
5. C. Oliveira, M. Wermelinger, J.L. Fiadeiro, A. Lopes. Modelling the GSM handover protocol in CommUnity. Electronic Notes in Theoretical Computer Science 141(3):3-25, 2005
6. A. K. Dey, Providing Architectural Support for Building Context-Aware Applications, PhD thesis, College of Computing, Georgia Institute of Technology, December 2000.
7. M. Sloman and E. Lupu, Engineering Policy-based Ubiquitous Systens, To appear in The Computer Journal, 2010.
8. Kephart, J. O. and Chess, D. M. 2003. The Vision of Autonomic Computing. *Computer* 36, 1 (Jan. 2003), 41-50.
9. A.K. Bandara, B. A. Price, B. Nuseibeh, Y. Rogers, A. Joinson, M. Sloman, E. C. Lupu, N. Dulay, A. Russo, Privacy Rights Management for Mobile Applications, In Proceedings of the 4th Symposium on Usable Privacy and Security, Pittsburg, USA, July 2008.
10. C.Mancini, K.Thomas, Y.Rogers, B.Price, L.Jedrzejczyk, A.Bandara, A.Joinson, and B. Nuseibeh, "From spaces to places: emerging contexts in mobile privacy," in Proceedings of the 11th International Conference on Ubiquitous Computing (UBICOMP 2009), 2009.
11. R. Wishart, D. Corapi, A. Madhavapeddy, M. Sloman, " Privacy Butler: A Personal Privacy Rights Manager for Online Presence," in proceedings of the 1st IEEE PerCom Workshop on Smart Environments (in press), Manheim, Germany, 29 Mar - 2 April 2010.
12. M. Langheinrich, Privacy by Design – Principles of Privacy-Aware Ubiquitous Systems**.** In Proceedings of the 3rd International Conference on Ubiquitous Computing (UbiComp 2001). LNCS No. 2201, Springer-Verlag, pp. 273-291, Atlanta, USA, 2001.
13. M. Mun, K. Shilton, K. Guan, G. Auyeung, N. Petersen and J. Burke. Personal Data Vault: A Privacy Architecture for Mobile Personal Sensing. UC Los Angeles: Center for Embedded Network Sensing. Retrieved from: http://escholarship.org/uc/item/5t09j13j, 21 Dec 2009.