Self-Organizing Infrastructures for Ambient Services

Klaus Herrmann

University of Stuttgart
Institute of Parallel and Distributed Systems (IPVS)
Universitätstr. 38, 70569 Stuttgart
klaus.herrmann@acm.org

Abstract The vision of Ambient Intelligence (AmI) as a new paradigm for supporting the mobile user in his daily activities is currently entering the focus of European research efforts. A high degree of autonomy on the part of the supporting software system is inherent to this vision of omnipresent and continuously running services. However, adequate concepts for creating respective infrastructures that may operate autonomously and in a self-organized fashion are still largely unexplored. We propose the *Ad hoc Service Grid* (ASG) as a dedicated AmI infrastructure that may be deployed in an ad hoc fashion at arbitrary medium-sized locations (shopping malls, construction sites, trade fairs, etc.). In this paper, we give an overview over our results thus far. We focus on the problems of service placement, discovery and lookup, and data consistency within an ASG environment and show how we have solved these problems with new self-organizing and adaptive algorithms. These vital functions are the basis for the realization of ASG systems and represent an essential contribution to AmI research in general.

1 Introduction

In recent years, the vision of *Ambient Intelligence* (AmI) [2] has produced considerable research efforts. The main idea of AmI is that mobile users may wirelessly access services (anywhere and anytime) that support them in their daily activities without putting any administrative burden on them. One essential building block of AmI are infrastructures that allow local access to local, facility-specific services. To enable services that enhance the user's interaction with his current environment, the physical surrounding of the user must be enriched with computing resources that enable service provisioning. One example scenario is a shopping mall that offers *ambient services* to customers, enabling them to navigate through the mall, find certain products quickly, and optimize the contents of their shopping cart, for example, according to the overall price or quality.

In this paper, we describe the *Ad hoc Service Grid* (ASG) infrastructure [5] that enables facility-specific ambient service provisioning at medium-sized locations (e.g., shopping malls, construction sites, trade fares, etc.). Our goal is to provide a service infrastructure that is easy to setup, flexibly scalable, and requires minimal administrative effort. The easy setup and the flexibility at the networking layer is achieved by adopting *Mobile Ad hoc Networking* (MANET) technology. To minimize the administration overhead, we propose a software layer that we call the ASG *Serviceware*. This software has the task of providing basic support for running services within an ASG in a self-organized fashion. That is, all aspects of executing services and adapting to changing

user demands shall be managed autonomously by the software. We will describe the core functions, algorithms, and protocols of the ASG Serviceware and show how they operate and interact in order to render the overall ASG self-organizing.

The rest of the paper is structured as follows. In Section 2, we discuss related work. The ASG model is introduced in Section 3 before we give an overview over the algorithms and protocols used to render its operation self-organizing in Section 4. In Section 5, we present our conclusions and give an outlook on future work.

2 Related Work

There are two major research strands in the area of AmI infrastructures at the moment. The first one deals with appropriate middleware systems. The research in the middleware area has been active for many years and is always quick in conquering new domains. Thus, a plethora of systems has been and continues to be proposed here. Cabri et al. propose the LAICA system as an agent-based middleware for AmI [1] and claim that agents "can naturally deal with dynamism, heterogeneity and unpredictability". Apart from that, no mechanisms are introduced that may substantiate this claim. The same is true for the SALSA system presented by Rodríguez et al. [11] for healthcare applications. Vallée et al. [14] seek to combine *multi-agent techniques* with *semantic web services* to create a system the can adapt to the user's current context in a context-aware service composition process. O'Hare et al. advocate the use of "agile agents" as a design principle for AmI [10]. These mobile agents are based on the *Beliefs Desires Intentions* (BDI) model that is well-known in intelligent agent research. However, no real motivation is given as to why this particular technology should be ideal for AmI.

The second strand originates from the area of *Service-Oriented Architectures* (SOA) and copes with the problems of finding, matching, and composing services in an AmI environment. Some variations of tools from the Web Services domain are being deployed for this purpose. Omnisphere is an architecture that supports the discovery of service components and the composition of higher-level services [12]. *Typed data flows* are used for service composition, and a matching mechanism is proposed that employs user preferences, devices capabilities, and the user's context to select the set of service components. Hellenschmidt et al. propose the SodaPop system that allows the composition of higher-level services from the components available on individual user devices [3]. They claim that their system enables devices to self-organize in order to collectively provide a service, but they do not substantiate this claim. Issarny et al. suggest to use a declarative language for specifying AmI systems [9]. The *Web Service Ambient Intelligence* WSAMI language allows the specification of composed services on the basis of Web Services technologies.

3 The Ad hoc Service Grid Model

3.1 Motivation

There are two obvious alternatives available for providing facility-specific, ambient services to mobile users. The first one is to use cellular phone networks as they are used

in a classical location-based service scenario. However, in this scenario, the available bandwidth is limited and any communication is expensive. The second alternative is the coverage with normal 802.11 (WLAN) access points which serve as a wireless extension of some wired infrastructure while services are being provided by some high-end server. This approach provides high bandwidths, and the communication is free of charge. However, it produces high costs for setting up the wired infrastructure. Moreover, it offers only limited flexibility and scalability since the wired infrastructure is fixed and cannot be easily changed.

We conclude that neither of the two technologies is particularly well-suited for the provisioning of facility-specific services. Therefore, we propose an alternative model that we explain in the following.

3.2 The Model

The ASG is based on the concept of *Service Cubes* (also called *Cubes* or *nodes* hereafter). A Service Cube is a PC-class computer that provides ad hoc networking capabilities and computing power. It has no peripheral devices (display or input devices), relies on a permanent power supply, and is equipped with a wireless network interface. In order to cover a given location with Ambient Services, a number of these Service Cubes is distributed over the location such that they can spontaneously set up a wireless network connecting all Cubes. Thanks to the concept of self-contained Service Cubes, an ASG network has a highly modular structure: New modules (Cubes) can be added and existing ones can be removed or repositioned to re-shape the network in an ad hoc fashion. This provides a flexible way of scaling an ASG to an appropriate size and allows for a quick and easy setup. No extensive planning and no construction work is necessary to cover a location in this way. It also enables different business models for providing an ASG since Service Cubes may be rented, collectively bought by different participants, or monolithically provided by some operator. Clients that wish to use ASG services may become a part of the ad hoc network by connecting to any of the Cubes with their mobile device. We assume the existence of some technology that allows for a seamless hand-over such that clients may move freely through the ASG network and always be connected to at least one Cube.

3.3 The Drop-and-Deploy Vision

The ultimate goal that we pursue with the ASG may best be characterized as *Drop-and-Deploy*: Anyone who decides to deploy an ASG simply has to distribute (drop) a certain number of Service Cubes at the respective location, switch them on, and *inject* the desired services at an arbitrary Cube. The structuring of the required software infrastructure, the binding of clients to services, and the adaptation to changing conditions is completely taken over by the ASG software. The road towards this goal holds some major challenges from diverse fields of computer science. In our work, we concentrate on a set of fundamental algorithms and protocols that are required in order to realize this vision of a self-organizing ASG infrastructure.

4 Algorithms and Protocols for a Self-Organizing ASG

Setting up a communication network is only the first step towards the operation of an ASG. In order to provide services within an ASG network, a software platform is required that can deal with the dynamics in the system. This *Serviceware* has to support the Drop-and-Deploy idea inherent to the ASG. In our work, we concentrate on three central aspects of such a Serviceware:

1. **Self-organizing service distribution:** Having a single replica of each service at some fixed Service Cube in an ASG may be sufficient to provide this service in a very basic way. However, if the ASG and the group of clients has a certain size, this solution is suboptimal. Temporary partitions in the network decrease the availability of the service. Requests have to be routed through the entire network, which wastes bandwidth, and increases response times. Therefore, it is vital to replicate services and to position them at specific Service Cubes such that most clients are served by a near-by replica. Moreover, this placement must not be static. It must be able to adapt if the client request patterns or the network topology change.
2. **Service discovery and lookup:** If services replicate and reposition dynamically, finding and using them becomes a challenge. Therefore, the idea of dynamic service placement directly implies the necessity for an adequate lookup service (LS) that is able to cope with this form of dynamics. This LS has to be distributed, too, and it has to apply some update strategy with an acceptably low overhead, such that updating the location information of repositioned replicas does not jam the network.
3. **Data consistency among stateful ASG services:** Most useful services that may be deployed in an ASG are stateful. That is, they store mutable data that may be read and written in a distributed way by clients. Therefore, an adequate consistency protocol is required that keeps the replicas of a service consistent with each other. This protocol, too, has to honor the specific conditions and the dynamics in an ASG.

In the following, we will examine each of these problems and our respective solutions in turn.

4.1 Distributed Service Placement

We have developed an adaptive service placement algorithm [6] that is run by each replica. This algorithm allows a replica to *migrate* from one node to another, to *replicate* (clone itself and subsequently migrate both clones), or to *dissolve* (remove itself from the system). The basic objective of this algorithm is to move replicas closer to the clients that use them. Additionally, the lookup service (cf. Section 4.2) enforces that a client always uses the service that is closest to it. These two principles define a feedback process: As a replica moves closer to its clients, it *attracts* even more clients from the respective area. This, in turn, increases the area's attractiveness for the replica which causes it to move closer, and so on. As the replica moves into the group of requesting clients, a negative feedback sets in and lets the replica converge to a stable position.

The placement algorithm itself is fully decentralized and requires no external control. It constantly inspects the incoming *message flows* and periodically takes a local

adaptation decision. This is possible without any additional communication between the replicas since they are coupled indirectly through the message flows they receive. This mechanism creates a stable, coordinated global placement that is adaptive to changes in the environment. The algorithm consists of three rules for the different adaptations:

1. **Idle Rule:** A replica dissolves (is removed) if it received less then α requests in a time interval m.
2. **Replication Rule:** A replica replicates to neighbor nodes u and w, if it receives a significant flow via u that consists of requests with an average path length of more than ρ hops. w is either set to the dominant node among the remaining neighbors, or the second replica stays at its current node. ρ is called the *replication radius*.
3. **Migration Rule:** A replica migrates to a neighbor node u, if the message flow that is coming in via u is stably *dominant* (larger than the sum of all remaining flows).

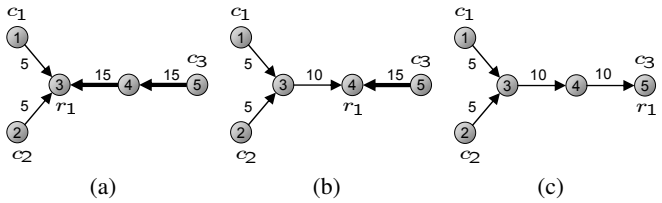

Figure 1. Exploiting dominating flows for incremental cost optimization.

The core adaptation algorithm invokes these rules in the order in which they are listed above and exits as soon as one rule *triggers* an adaptation. The idle rule simply garbage-collects unused replicas and enforces a constant refreshment. This avoids that the system gets permanently stuck in some suboptimal configuration. The replication rule applies a pressure on the system such that replications happen until each replica covers at most a network area of radius ρ. This creates an overall number of replicas that depends on ρ and on the diameter of the network, and effectively leads to the division of the network into cells. The migration rule forces replicas to move to locations where the magnitudes of the incoming message flows are in balance. Figure 1 shows how the overall message flow (numbers at the edges) is reduced by migrating towards a dominant flow. This global behavior is not directly coded into the rules. It *emerges* as a number of replicas apply the rules and interact indirectly with one another.

4.2 Service Discovery and Lookup

The instances of the ASG lookup service (LS) [7] are distributed over an ASG network. An ASG network is clustered, and each ordinary node has at least one cluster head in its direct neighborhood. Cluster heads are used to run base services (like the LS) in the ASG. Each LS instance holds location information for all active service replicas. Due to the distributed nature, updates have to be propagated among the LS instances. Since

replica migrations may be executed frequently, and since the transmission bandwidth in the wireless network is the most important resource in the ASG, we refrain from using flooding updates. Instead, we employ a lazy propagation strategy.

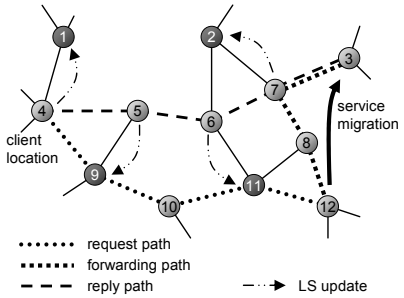

Figure 2. Request-driven update process.

Figure 2 depicts the update process. Only those messages are flooded that announce the creation or the removal of a replica. All other updates (when a replica migrates) are piggybacked by normal service reply messages. Nodes inspect each message that they relay. If a node finds a piggybacked LS update, it informs its cluster head who applies the update to its service table. In this way, updates are propagated *lazily* in those regions that are involved in the sending or the relaying of requests. All other regions remain outdated. However, this does not void the validity of the lookup information due to a second mechanism that cooperates with this lookup process: Every time a replica migrates, it leaves a *forward pointer* at its old location. If a request arrives at this location, it is forwarded, possibly over several former locations of the replica, until it arrives at its current node. The replica sends a reply together with an LS update back to the client. This mechanism ensures that even outdated lookup information is still sufficient to deliver a request correctly. The lazy updating mechanism requires minimal message overhead since updates are only made if needed, and only one dedicated update message is sent to each LS instance along the reply paths.

4.3 Data Consistency

Due to the possible dynamics in an ASG, preserving the consistency in a group of replicas requires an optimistic approach. Replicas may be temporarily separated due to network partitions, or they may be spontaneously created or removed. In order to cope with these circumstances, we have extended the well-known Bayou anti-entropy protocol [13]. The new *Bounded Divergence Group Anti-Entropy Protocol* (BD-GAP) can run reconciliation processes among an arbitrary group of replicas, ensuring eventual consistency. Each replica may autonomously start a reconciliation process and synchronize with the fellow replicas currently known to it. It gets this information from the lookup service. Conflicts due to concurrent initiations of reconciliations are resolved automatically. The protocol chooses the order in which to exchange updates with other

replicas such that the amount of data that has to be exchanged is minimized. Furthermore, it limits the degree to which data stores diverge in such a way that complete state transfers, as they are necessary in Bayou, are completely avoided.

The BD-GAP introduces a stronger coupling than the original anti-entropy protocol by exploiting the nature of the ASG. But still, it allows for enough decoupling to preserve the increased availability introduced by the original protocol. It is adaptive to the degree of dynamics in the system since it gradually reverts to the original pair-wise protocol when the dynamics in the system increases. In situations with low dynamics, it exploits the fact that most replicas are accessible in order to do reconciliations more thoroughly which increases the overall consistency.

4.4 Additional Results

In addition to the results highlighted above, we have also designed and implemented `MESH`*`Mdl`* [4], a core middleware on top of which our algorithms and protocols are implemented. `MESH`*`Mdl`* is based on mobile agents and the tuple space paradigm to allow for decoupled and asynchronous communication.

In order to show why and in which way the proposed solutions are self-organizing, we created a novel model for *Self-Organizing Software Systems* (SOSS) [8]. The SOSS model can be used to classify existing software systems in order to show whether they are in the class of SOSS or not. Such a model did not exist before which has lead to a growing confusion about the nature of existing systems. We envision that it will serve as an ordering tool beyond the scope of our concrete work, and that the classification of software systems will provide new insights into their general nature.

5 Conclusions and Future Work

In our work, we have made the first step towards a self-organizing infrastructure for ambient services. The three mechanisms that we presented above lay the foundation for this development by providing a self-contained set of functionalities that is required by AmI environments in order to offer facility-specific services to mobile users. We have shown how a self-organized service placement, a self-organized lookup system, and an adequate consistency protocol can be modeled. Furthermore, we have implemented these concepts on top of a set of simple middleware abstractions to proof their validity. Our SOSS model allows us to argue precisely why and how the resulting system is self-organizing.

In the future, more sophisticated systems can be built on these fundamental mechanisms. For example, we have not touched the topics of security and privacy in an ASG environment. These are both essential ingredients to commercially exploitable systems. Moreover, the extension of the ASG with gateways to the Internet and the federation of multiple ASG remain open issues that will be in the focus of our future work.

References

1. G. Cabri, L. Ferrari, L. Leonardi, and F. Zambonelli. The LAICA project: supporting ambient intelligence via agents and ad-hoc middleware. In *Proceedings of the 14th IEEE Inter-*

national Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise, pages 39–44, June 2005.

2. K. Ducatel, M. Bogdanowicz, F. Scapolo, J. Leijten, and J.-C. Burgelman. Scenarios for Ambient Intelligence in 2010. Technical Report, The IST Advisory Group (ISTAG), 2001.
3. Michael Hellenschmidt. Distributed Implementation of a Self-Organizing Appliance Middleware. In Gérard Bailly, editor, *Proceedings of sOc-EUSAI 2005 (Smart Objects Conference)*, pages 201–206, 2005.
4. Klaus Herrmann. MESHMdl – A Middleware for Self-Organization in Ad hoc Networks. In *Proceedings of the 1st International Workshop on Mobile Distributed Computing (MDC'03)*, May 2003.
5. Klaus Herrmann. *Self-Organizing Infrastructures for Ambient Services*. PhD thesis, Berlin University of Technology, July 2006. (publication pending).
6. Klaus Herrmann, Kurt Geihs, and Gero Mühl. Ad hoc Service Grid – A Self-Organizing Infrastructure for Mobile Commerce. In *Proceedings of the IFIP TC8 Working Conference on Mobile Information Systems (MOBIS 2004)*. IFIP – International Federation for Information Processing, Springer-Verlag, September 2004.
7. Klaus Herrmann, Gero Mühl, and Michael A. Jaeger. A Self-Organizing Lookup Service for Dynamic Ambient Services. In *25th International Conference on Distributed Computing Systems (ICDCS 2005)*, pages 707–716, Piscataway, NJ, USA, June 2005. IEEE Computer Society Press.
8. Klaus Herrmann, Matthias Werner, and Gero Mühl. A Methodology for Classifying Self-Organizing Software Systems. In *International Conference on Self-Organization and Autonomous Systems in Computing and Communications (SOAS'2006)*, September 2006.
9. Valérie Issarny, Daniele Sacchetti, Ferda Tartanoglu, Françoise Sailhan, Rafik Chibout, Nicole Lévy, and Angel Talamona. Developing Ambient Intelligence Systems: A Solution based on Web Services. *Automated Software Engineering*, 12(1):101–137, 2005.
10. Gregory M. P. O'Hare, Michael J. O'Grady, Rem W. Collier, Stephen Keegan, Donal O'Kane, Richard Tynan, and David Marsh. Ambient Intelligence Through Agile Agents. In Yang Cai, editor, *Ambient Intelligence for Scientific Discovery*, volume 3345 of *Lecture Notes in Computer Science*, pages 286–310. Springer-Verlag, 2004.
11. Marcela Rodríguez, Jesus Favela, Alfredo Preciado, and Aurora Vizcaíno. An Agent Middleware for Supporting Ambient Intelligence for Healthcare. In *Second Workshop on Agents Applied in Health Care (ECAI 2004)*, August 2004.
12. F. Rousseau, J. Oprescu, L.-S. Paun, and A. Duda. Omnisphere: A Personal Communication Environment. In *Proceedings of the 36th Hawaii International Conference on System Sciences (HICSS-36 2003)*, 2003.
13. D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing Update Conflicts in Bayou, a Weakly Connected Replicated Storage System. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 172–182, New York, NY, USA, 1995. ACM Press.
14. M. Vallée, F. Ramparany, and L. Vercouter. A Multi-Agent System for Dynamic Service Composition in Ambient Intelligence Environments. In *Advances in Pervasive Computing, Adjunct Proceedings of the Third International Conference on Pervasive Computing (Pervasive 2005)*, May 2005.