Noname manuscript No.
(will be inserted by the editor)

Jeppe Brønsted · Klaus Marius Hansen · Mads Ingstrup

A Survey of Service Composition Mechanisms in Ubiquitous Computing

the date of receipt and acceptance should be inserted later

Abstract Composition of services, i.e., providing new services by combining existing ones, is a pervasive idea in ubiquitous computing. We surveyed the field by looking at what features are actually present in technologies that support service composition in some form. Condensing this into a list of features allowed us to discuss the qualitative merits and drawbacks of various approaches to service composition, focusing in particular on usability, adaptability and efficiency. Moreover, we found that further research is needed into quality-of-service assurance of composites and into contingency management for composites—one of the concerns differentiating service composition in ubiquitous computing from its counterpart in less dynamic settings.

1 Introduction

The ability to seamlessly compose the services from various devices in a more or less ad-hoc manner is a frequently emphasised feature of ubiquitous computing (cf. e.g. [19–21]). Given the abundant mention of this feature in visions and scenarios, one should expect a sizeable body of research and development work dedicated to its realisation. However, in practise we have found far fewer cases than expected where service composition for ubiquitous computing has been a main focus of research efforts. By "main focus" we mean that actually designing, implementing and evaluating a composition technology has been a main part of the contribution of the work. With that said, however, work has been carried out in the area. This paper presents an initial survey of it and draws out the experiences and lessons it holds for continued development of the field. In particular, there are several reasons that prompted us to do this work.

Firstly, we believe, with our colleagues whose work we survey, that there is a real need for good composition mechanisms. This need arises on a variety of temporal scales, including, for instance, the user on fieldwork who is prompted by immediate circumstance to compose a set of services that allows a display on a GPS device to temporarily replace a broken one on an otherwise functional cell-phone. Or the administrator who needs to tailor commercially acquired services to the needs of people in his or her particular organisation. Towards the goal of developing a good service composition mechanism, our contribution is to help understand what "good" might mean when talking about service composition for ubiquitous computing, or rather to give an account of advantages and disadvantages of various approaches and identify areas that need further attention from the research community.

Secondly, we wish to support the claim of service composition for ubiquitous computing as an area of study in itself. Composition is a natural concept around which to structure ubiquitous systems and individuate parts of it that are suitably subject to e.g. contingency management, application, or storing, as witnessed by its frequent use as a new and more open-ended replacement for the traditional application concept of desktop computers. This makes it a viable topic of research in itself.

The paper is organised as follows. In section 2 we describe the approach we have used to arrive at the categorisation of the surveyed composition mechanisms used to compare their features in table 1. Based on this comparison, we move on to a more qualitative discussion in section 3, before concluding in section 4.

2 Service composition categorisation

We only consider architectures that enable composition of services. The following definition of a service is used to decide which technologies we include in the survey:

> A *service* is a unit of runtime software that is accessible by others

This allows for variation in the rationales of service definitions used in the surveyed papers. A further constraint is that the technologies should be targeted towards ubiquitous computing. As a minimum, this means the range of devices the

Department of Computer Science, University of Aarhus
Aabogade 34, 8200 Aarhus M, Denmark
E-mail: {jb,klaus.m.hansen,ingstrup}@daimi.au.dk

technology works with is not limited to desktop PCs, and that it allows for distributed deployment of the composed services. Finally, we take composition to be a grouping of services interacting with a certain purpose.

In the following, we outline our categorisation of service composition features and give examples of middleware that implements those features. We do not propose a rigorous, would-be normative definition of which features are of relevance to service composition in ubiquitous computing. Rather we studied features what are present in systems that can be said to support service composition for ubiquitous computing.

The overall purpose of the survey is to compare and position each architecture relative to the others in terms of the quality trade-offs they engender. In order to enable this we examine what features each composition mechanism has within a select set of concerns.

The categorisation framework presented in this section is divided into three main parts that have been selected to inform the discussion on qualitative trade-offs, and to expose variation points in the technologies surveyed. The first part concerns by who, when, and how service composition is specified. The second part deals with runtime properties of the technologies and the third part concerns how the composite is deployed and how it has been evaluated. In table 1, the categorisation framework is applied to the surveyed technologies. In the following sections, the first word in italics describes a column in table 1.

2.1 Specification

Specified by The first step in establishing service composition is to specify what should be composed and how. The mechanism used for this can be aimed at developers as a tool for constructing general applications, or it can be aimed at the user to let him tailor an application for his particular needs. If the developer specifies the composite he can only try to anticipate what services will be available at runtime, whereas if the user specifies the composite he can design the composite from the available services and compose services and devices in new unanticipated ways as in [5, 21].

In Service Composition for Mobile Environments (SCfME) [3] which encompasses protocols for service composition in mobile ad hoc network environments, the composition is specified by application developers through a DAML description [1] of a "Description-Level Service Flow" (DSF) in which sequences of services may be described.

Another example is ICrafter [19] in which users may combine services from different devices and have an aggregated user interface generated.

With UbiDev [14] an application developer supplies an ontology, classifiers, and user interfaces for services in an application. The classifiers map resources on devices into concepts in the ontology. A service is then defined as an atomic action that transforms an input resource yielding a new resource as output. Services are constrained by the concept (from the ontology) that it accepts as argument, and the concept it produces as output.

Specified at The composite can be specified at development time before the composition framework is up and running or at runtime. Note that if the composite is specified before runtime, it implies that it is done by a developer since the user, at that time, has no way of interacting with the system. If the composite can be specified at runtime, there is no need for restarting the composition framework.

Both ICrafter and SCfME provide for runtime composition specification. In ICrafter, users may explore services available at runtime through a general purpose Interface Manager (IM). With the interface of the IM the available services can be explored and a subset selected. An aggregate interface is then requested for the selected set of services. In SCfME an application developer may supply new DSFs at runtime and have these bound dynamically to an "Execution-Level Service Flow" (ESF) containing references to actual services.

UbiDev is more static in the sense that an ontology and corresponding classifiers need to be provided prior to deploying the application. To change the possible service compositions, either the ontology or the classifiers need to be changed, something that appears to be only possible at development time.

Specified in The specification can be specified by providing source code, configurations, or interacting with a tool. If the composite is specified by user interaction, an underlying representation has to be available to make the specification persistent.

In ICrafter, the composition is specified by interacting with a tool in which services are rendered. Services are described with a SDL (Service Description Language) that includes a simple type system oriented towards UI generation.

In SCfME, the service composition is described via a configuration (a DAML XML document) and in UbiDev the specification is partly programmatic since application developers need to supply classifiers.

Level The services in the composition specification can be represented as instances, types, or implicitly. In the case of instances, a particular service is bound to the composite by e.g. an URN [16]. If the services are specified by types there can be several candidates for each service specified. Finally, if the user instead of specifying how services are connected, requests a task from the framework that has to be resolved into a composition of services, we regard the services as being specified implicitly.

With respect to the level of description, SCfME is characteristic of an approach in which the composition is described at a type level and resolved at runtime by the service composition implementation to an instance level. ICrafter, on the other hand, operates only on an instance level in that

it is specific services that users select. Finally, UbiDev operates on an implicit level based on the ontology and classifiers specified. One example is a generic messaging service, *document_to_display* that can be adapted to the context by UbiDev. If the user context is a personal phone, UbiDev will compose the services *ascii_to_wav*, *wav_to_adpcm*, and *adpcm_to_voice* to provide the *document_to_display* service.

Quality of Service (QoS) A few of the technologies have support for specifying quality of service requirements in the composite specification. If, e.g., the services are specified as types the selection of the actual instances can be guided by the quality of service specification.

In the Amigo service composition mechanism [22] services are described as semantic Web services (using OWL-S [15]) in which atomic processes have QoS attributes with values obtained from runtime measurements. An example of a QoS requirement would be that latency should be less a threshold. At runtime, an abstract specification of a composite service is matched against possible realisations, the latencies for the realisations are calculated, and the best composition is chosen (also subject to other constraints) [17].

2.2 Runtime

Contingencies The technologies surveyed provide different levels of support for contingency handling. The trivial option is to have no support for contingencies at all. A slightly more advanced solution is to detect the contingency and alert the user, and a natural extension of this approach is to resolve the contingency automatically. It should be noted that in some cases, it is impossible to resolve contingencies automatically and in these situations, the user should be in the loop.

As an example, in SCfME the state of the execution of a composition is check-pointed by sending partial results back to the service requester. If the Composition Manager fails, the service requester is notified and may create a new concrete ESF based on the original abstract DSF and its latest checkpoint and finds a new composition manager.

In UbiDev, since the composition is implicit and dynamic, contingencies such as partial failures will be handled automatically as classifiers find new ways of realising compositions.

Resource use Since many devices for ubiquitous computing have limited resources its relevant to look at resource use. Since its not feasible to make comparable measurements for all the technologies, we have made a rough estimate of what kind of system the composition mechanism requires. One type is a server platform and another is a typical PDA. A third category would be a smaller sensor/actuator platform such as Motes (`http://www.tinyos.net/`).

A number of solutions apply semantic Web technology (e.g., SCfME and Amigo) which means that at least parts of the middleware will not scale to low-end devices. Other than that most service composition mechanisms seem to aim at PDA-type device and none focuses on sensor/actuator platforms.

Scalability As with resource use we only provide an estimate of how the composition mechanism scales.

In Amigo, a part of the service composition mechanism was to build a "global" automaton. It is not clear what the scope of that automaton is, but together with the use of OWL-S this means that the scalability of Amigo is low.

All devices in one.world run the one.world system platform. Services are composed in a decentralised manner and thus no node has the sole responsibility of the composite. This makes it possible to create composite services consisting of arbitrarily many services.

2.3 Deployment

Infrastructure In ubiquitous computing environments services might not be connected at all times. Devices may enter and leave the network and therefore it is important whether the composite has to have a fixed connection during operation or whether devices can enter and leave on an ad hoc basis.

In SCfME devices are peers and communicate via ad hoc protocols. In particular, a Group-based Service Routing (GSR) on-demand protocol in which the route is constructed during service discovery is used for routing service invocations.

In contrast, ICrafter assumes a fixed infrastructure in which a global communication infrastructure which resembles a tuplespace may be constructed.

Topology Some of the mechanisms require that a central node acts as a coordinator for the composite and thereby imposing a centralised structure in the composite, whereas other mechanisms allows for a decentralised structure.

SCfME is decentralised in that after an ESF is created, any node in the system can be used as a Composition Manager that handles the execution of the composition. The Composition Manager may be dynamically changed during the execution.

Based on finite state automata for individual OWL-S processes, the Amigo service composition mechanism builds a global finite state automaton for all services. In this sense, the Amigo service composition mechanism relies on a central component.

Evaluation Most of the technologies presented are evaluated by demonstrating that the composition mechanism can be used to implement various ubiquitous computing applications. *Example* applications invented by applications developers typically demonstrate a wide range of features whereas application *scenario*s developed in cooperation with users, have emphasis on posing relevant challenges. In

some of the projects, it is evaluated how usable the composition mechanism is for users and/or developers, and, finally, some projects present measurements of various performance parameters.

In SpeakEasy, an example application, *Casca*, supports collaborative work by letting users share files, devices, and services on an ad-hoc basis, and thus demonstrates various features of the composition mechanism. Usability is evaluated by observing users interacting with the SpeakEasy service browser to accomplish a task.

The PalCom service composition mechanism have been used to implement a set of application scenarios that have been developed in cooperation with end-users using participatory design techniques. The scenarios range from enabling landscape architects to visualise digital data in the context of the physical world to supporting rehabilitation of children with Downs syndrome.

3 Qualitative aspects of service composition

In the following we discuss composition mechanisms from an architectural perspective taking the categorisation presented as an outset. In doing so, we look at (architectural) software qualities [12] of the composition mechanisms that are of particular relevance to ubiquitous computing.

3.1 Usability

Usability is defined in one the predominant quality frameworks as "concerned with how easy it is for the user to accomplish a desired task and the kind of user support the system provides." [2, p. 90]. This includes understandability, learnability, operability, and attractiveness [12].

Although the traditional view is that usability is first and foremost determined by the user interface of systems, there are several ways in which the architecture may strongly influence the usability of a system, e.g. the support for undo functionality [2]. Similarly, the service composition mechanism influences usability in a number of ways.

Our main criteria for assessing the usability of a composition mechanism is whether the composite services built with a given mechanism suit the users optimally given what services are available for composition. A strong determinant of this is how the composite services come to exist. Another is whether a composite, once created, can adapt (or be adapted) to change of circumstance, such as due to contingent availability of services and resources or malleability of users' intentions with the composition. We discuss the first issue below and the second in the next subsection.

Regarding the first, we may question whether developer-specified, users-specified, or e.g. automatic synthesis based on interpretation of high-level sentences [7] leaves the user better off. The PalCom assembly concept is based on an elaborate conception of this issue. It is held that autonomic,

Table 1 Categorization of Service Composition Mechanisms

Technology	Composition Specification					Runtime			Deployment		
	Specified by	Specified at	Specified in	Level	QoS	Contingencies	Resource use	Scalability	Infrastructure	Topology	Evaluation
Amigo [17, 22]	End-users	Runtime	End-User int.	Implicit	Runtime	Automatic	PDA+Server	Low	Fixed	Centralized	Sce. perf.
Aura [8]	End-users	Runtime	Configuration	Types	Runtime	Automatic	PDA+Server	Unknown	Ad Hoc	Centralized	Example
Daidalos [23]	Unknown	Runtime	Configuration	Types	Runtime	Runtime	Unknown	Average	Unknown	Centralized	Example
GAIA [20]	App. Devs	Dev. time	Configuration	Types	None	Automatic	PDA+Server	Unknown	Fixed	Centralized	Example
ICrafter [19]	End-users	Runtime	End-User int.	Instances	None	None	PDA	Average	Fixed	Centralized	Example
Obje [5]	End-users	Runtime	End-User int.	Instances	None	Unknown	PDA	Unknown	Ad Hoc	Decentralized	Example
one.world [9, 10]	App. Dev	Runtime	Source Code	Instances	None	Detection	J2SE	High	Ad Hoc	Decentralized	Sce., Usab., Perf.
PalCom [21]	End-users	Runtime	Configuration	Instances	None	Automatic	PDA	Low	Ad hoc	Centralized	Scenario
Paths [13]	App. Devs	Runtime	Configuration	Implicit	None	None	PDA	Average	Fixed	Centralized	Scenario
SCIME [3]	App. Devs	Runtime	Configuration	Types	None	Detection	PDA [a]	Low	Ad hoc	Decentralized	Performance
SpeakEasy [4, 6, 18]	End-users	Runtime	End-User int.	Instances	None	Detection	PDA	High	Ad Hoc	Decentralized	Ex., Usab.
UbiDev [14]	App. Devs	Dev. time	Source Code	Implicit	None	Automatic	PDA+Server	Low	Ad Hoc	Centralized	Example

[a] However, Composition Managers need to be able to handle DAML

perhaps AI-based, composition or developer-specified composition based on anticipation of the user's needs are fine in so far as the resulting composite does what the user wants it to. However these approaches are for a variety of reasons not flawless and complete enough to work consistently and so they need to be supplemented by architecture and tool support that empowers the user to create composites or modify existing ones to suit their purpose. [11]

Thus the way a composite service is specified is influential on usability by (1) deciding whether the users can specify composite services themselves, (2) whether they can do so at runtime and (3) how it is done. Only 5 of the 12 surveyed technologies enable end-users to specify composite services[1]. Non-support for end user composition appears in several cases to be due to the focus of the research project in question rather than a fundamental constraint of the architecture it produces: In 3 of the remaining 7 composition mechanisms the composite is specified in a configuration file and at runtime. These are arguably only a composition tool away from supporting end-user composition, albeit the configuration file format may be more or less constraining depending on its level of support for scripting.

In general, the considered composition mechanisms may be geared more towards the user or the developer. In ICrafter, for instance, the composition is explicit mainly in the tools used by the user, rather than at the architectural level. Speakeasy, in contrast, does not provide tools for the end user to do composition, but consists of a more general framework that developers can use in their applications. Concerning operability we must assume users have a greater range of tasks they can perform if the composition mechanism allows them to specify new composites, and that this also allows for greater adaptability in the short term.

3.2 Adaptability

Adaptability is the "capability of the software product to be adapted for different specified environments without applying actions or means other than those provided for this purpose for the software considered" [12, p. 11]. (In [2, p. 80] this is related to runtime 'modifiability').

The majority of the surveyed composition mechanisms (9 out of 12) allow service composition to be specified at runtime. Furthermore, most mechanisms do service discovery and coordination based on specification of needed types of services. This all supports adaptability which is arguably a central quality of ubiquitous computing systems that are often characterised as being able to work in a heterogeneous, changing system context. SCfME, e.g., has the possibility of providing a high-level (type-based) description of a needed service at runtime.

The three dimensions of *specification* imply different temporal scales at which adaptation can be supported. Source-code and developer based specification that happens at development time necessitates a much longer turn-around time for adaptations than do end-users operating at runtime.

On the other hand, adaptability is less well supported for contingencies such as services disappearing or failing. An example, though, is UbiDev in which the Service Manager is responsible for rebinding services upon partial failure.

Another aspect of adaptability is in deployment where supporting an ad hoc infrastructure in a decentralised topology has the potential of being most adaptive. Obje, one.world, SCfME, and SpeakEasy all support this combination whereas none of these have automatic contingency management.

3.3 Efficiency

Efficiency is the "capability of the software product to provide appropriate performance, relative to the amount of resources used, under stated conditions" [12, p. 10]. This pertains to both time behaviour and resource utilisation such as memory use. Again, this is arguably a central quality of ubiquitous computing in that embedded systems often play a central role.

It is characteristic that none of the surveyed appear to scale to low-end devices such as sensors even though this may be relevant. In addition, few of the composition mechanisms are highly scalable with one.world and SpeakEasy as exceptions.

4 Conclusions and Future Work

This paper has presented an initial survey of service composition in ubiquitous computing. Quite a few service composition mechanisms exist of which we have only reported on a few here. In doing so, however, it has been characteristic that no papers have focused mainly on service composition. The characteristics that we have outlined show that there are indeed opportunities for doing so.

In particular, service composition is entangled with several complex features such as service discovery and matching, contingency management, and reconfiguration and therefore provides a good frame around which to explore those features and their interrelation. This, however, is an area where further research is needed: the surveyed technologies only begin to scratch the surface of contingency management and efficient resource management. Likewise, more work is needed to explore the relationship between manual and autonomic execution of functionality. Concerning the notion of service composition, we found very limited signs of research that leveraged the result of previous research in the area. Such leverage is arguably easier to achieve with a common conceptualisation of the research area, and we consider this survey a step towards that in addition to providing an overview.

The obvious further work on this survey is to make it more complete in several ways. One is by including further

[1] PalCom Assemblies, listed in table 1 as being specified in a configuration file, has at least one tool for allowing users to compose services (it uses the parsed configuration file as a living representation, and generate a file one when it is saved)

examples of relevant ubiquitous computing systems. Another is by considering other forms of service composition (such as the ones used for web services) and discuss their relevance to ubiquitous computing. Both of these directions we believe will benefit the development of quality service composition mechanisms in ubiquitous computing.

Acknowledgements

The research presented in this paper has been partly funded by the EU projects "PalCom" (IST-002057; `http://www.ist-palcom.org`) and "Hydra" (IST-034891; `http://www.hydra.eu.com`).

References

1. ANKOLEKAR, A. ET AL. DAML-S: Web service description for the Semantic Web. In *Proceedings of ISWC '02* (2002), vol. 2342 of *LNCS*, Springer-Verlag, pp. 348–363.
2. BASS, L., CLEMENTS, P., AND KAZMAN, R. *Software Architecture in Practise*, 2nd ed. Addison-Wesley, 2003.
3. CHAKRABORTY, D., JOSHI, A., FININ, T., AND YESHA, Y. Service composition for mobile environment. *Mobile Networks and Applications 4*, 10 (August 2005), 435–451.
4. EDWARDS, W. K., NEWMAN, M. W., SEDIVY, J., AND IZADI, S. Challenge: recombinant computing and the speakeasy approach. In *Proceedings of ACM MobiCom* (New York, USA, 2002), ACM Press, pp. 279–286.
5. EDWARDS, W. K., NEWMAN, M. W., SEDIVY, J. Z., AND SMITH, T. F. Bringing network effects to pervasive spaces. *IEEE Pervasive Computing 4*, 3 (2005), 15–17.
6. EDWARDS, K., ET AL. Using speakeasy for ad hoc peer-to-peer collaboration. In *Proceedings of ACM CSCW '02* (2002), ACM Press, pp. 256–265.
7. FUJII, K., AND SUDA, T. Dynamic service composition using semantic information. In *Proceedings of the 2nd International Conference on Service Oriented Computing* (New York, New York, U.S.A, Nov. 2004), ACM.
8. GARLAN, D., SIEWIOREK, D., SMAILAGIC, A., AND STEENKISTE, P. Project Aura: toward distraction-free pervasive computing. *Pervasive Computing, IEEE 1*, 2 (2002), 22–31.
9. GRIMM, R. One. world: Experiences with a Pervasive Computing Architecture. *IEEE Pervasive Computing 3*, 3 (2004), 22–30.
10. GRIMM, R., DAVIS, J., LEMAR, E., MACBETH, A., SWANSON, S., ANDERSON, T., BERSHAD, B., BORRIELLO, G., GRIBBLE, S., AND WETHERALL, D. System support for pervasive applications. *ACM Trans. Comput. Syst. 22*, 4 (2004), 421–486.
11. INGSTRUP, M. *Towards distributed declarative architectural reflection*. PhD thesis, University of Aarhus, 2006.
12. ISO/IEC. Software engineering—product quality, part 1–4, 2001. ISO-9126-1,-2,,3,-4.
13. KICIMAN, E., AND FOX, A. Using dynamic mediation to integrate COTS entities in a ubiquitous computing environment. In *Proceedings of HUC2000* (2000), no. 1927 in LNCS, pp. 211–226.
14. MAFFIOLETTI, S., KOUADRI, M., AND HIRSBRUNNER, B. Automatic resource and service management for ubiquitous computing environments. *Pervasive Computing and Communications Workshops, 2004. Proceedings of the Second IEEE Annual Conference on* (2004), 219–223.
15. MARTIN, D., ET AL. Bringing Semantics to Web Services: The OWL-S Approach. *Proceedings of the First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004)* (2004), 6–9.
16. MOATS, R. URN Syntax. RFC 2141 (Proposed Standard), May 1997.
17. MOKHTAR, S., LIU, J., GEORGANTAS, N., AND ISSARNY, V. QoS-aware dynamic service composition in ambient intelligence environments. In *Proceedings of IEEE ASE* (2005), ACM Press New York, NY, USA, pp. 317–320.
18. NEWMAN, M., ET AL. Designing for serendipity: supporting end-user configuration of ubiquitous computing environments. In *Proceedings of DIS '02* (New York, NY, USA, 2002), ACM Press, pp. 147–156.
19. PONNEKANTI, S., LEE, B., FOX, A., HANRAHAN, P., AND WINOGRAD, T. ICrafter: A Service Framework for Ubiquitous Computing Environments. *Proceedings of Ubicomp 1* (2001).
20. ROMÁN, M., ET AL. Dynamic application composition: Customizing the behavior of an active space. In *Proceedings of IEEE PERCOM* (Washington, DC, USA, 2003), IEEE Computer Society, p. 169.
21. SVENSSON, D., HEDIN, G., AND MAGNUSSON, B. Pervasive applications through scripted assemblies of services. In *Proceedings of 1st International Workshop on Software Engineering of Pervasive Services* (2006).
22. VALLEE, M., RAMPARANY, F., AND VERCOUTER, L. Flexible Composition of Smart Device Services. In *The 2005 International Conference on Pervasive Systems and Computing (PSC-05), Las Vegas, Nevada, USA., June* (2005), pp. 27–30.
23. YANG, Y., MAHON, F., WILLIAMS, M. H., AND PFEIFER, T. Context-aware dynamic personalised service re-composition in a pervasive service environment. In *Proceedings of Ubiquitous Intelligence and Computing* (2006), vol. 4159 of *LNCS*, Springer, pp. 724–735.