This is the author version of the paper. Original at: O. Davidyuk, I. Sanchez, E. Gilman and J. Riekki "An Overview of Interactive Application Composition Approaches", Open Computer Science. Volume 5, Issue 1, ISSN (Online) 2299-1093, DOI: 10.1515/comp-2015-0007, December 2015.

Review Article **Open Access**

Oleg Davidyuk*, Iván Sánchez Milara, Ekaterina Gilman, and Jukka Riekki

An Overview of Interactive Application Composition Approaches

Abstract: Application composition is an approach to create applications by using software components as building blocks. Applications can be composed of Web Services and resources associated with mobile devices, displays and various augmented everyday objects. In this article, we focus on ubiquitous applications which aim at supporting users' needs and everyday activities. Application composition is particularly suitable for these applications as it proposes to compose ubiquitous applications by choosing the appropriate set of services and resources and their configuration as required by users, their needs and other contexts. This article gives an overview and classifies interactive application composition approaches. These approaches provide the necessary user tools and various user interfaces to enable users themselves to specify their needs and achieve their goals with composed applications. The approaches in this article are analyzed according to the user support provided during the application life-cycle and user involvement during the application composition and execution phases. Furthermore, we look inside the design of user interfaces for visual and non-visual user tools and discuss their advantages and disadvantages. In addition to giving an overview of this research field, our aim is to provide means for describing, classifying and comparing different composition approaches.

Keywords: Ubiquitous computing interaction design application composition

DOI DOI
Received ..; revised ..; accepted ...

***Corresponding Author: Oleg Davidyuk:** Department of Computer Science and Engineering. University of Oulu, Pentti kaiteran katu 1,PO Box 4500, FI-90014 Oulu, Finland, E-mail: oleg.davidyuk@gmail.com

Iván Sánchez Milara, Ekaterina Gilman, Jukka Riekki: Center for Ubiquitous Computing. University of Oulu, Pentti kaiteran katu 1,PO Box 4500, FI-90014 Oulu, Finland, E-mail: firstname.lastname@ee.oulu.fi

1 Introduction

Software composition is a widely used software engineering method for constructing and maintaining complex software structures using small software parts as building blocks. These parts are accessible as runtime units of software, such as components and services. Software composition aims to reduce the complexity of the design and to facilitate the construction of software systems by reusing software parts as much as possible. Those parts are put together in order to meet the requirements imposed, among others, by users, software developers and smart environments.

Software composition was initially presented by McIlroy [1] in 1969. In the last decade, this research field has resurfaced due to the materialization of Ubiquitous Computing[2]. Applications running in computing enhanced environments must be linked to multiple resources in order to be capable of assisting and interacting with users. These applications provide a different kind of service to users, understanding by service the capabilities of a resource (e.g. a I/O device, a computation or storage) that can be accessed through some software interface.

Application composition, i.e. the approach to construct applications by composing and configuring the appropriate set of resources and services either by the users themselves, or by the system based on users' needs and other contexts, is specifically useful for creating ubiquitous applications. By composing services across multiple resources, a collection of resources may provide a seamless application functionality that would otherwise not be available. For instance, application composition can be utilized to build powerful virtual devices by grouping the capabilities of resource-limited ones [3], to build multimodal user interfaces by combining and controlling inputs and outputs from multiple resources [4] and to develop applications that can dynamically adapt to changes in the environment [5, 6]. As a ultimate goal, application composition aims to support users' needs and activities by choosing the adequate set of services and configurations which together pro-

vide required functionality. In such a way, application composition has been used in the fields of rehabilitation [7], elderly supervision [8], learning [9], home and office automation[10–12], and task facilitation [13–16].

The literature presents two different approaches for application composition: automated composition and interactive application composition. Both differ in the degree of user involvement and the degree of control. Automated composition approaches (present mainly in task-based computing research [14, 17]) assume that tasks such as application configuration, management and provisioning should be performed by the system. Users concentrate only on interacting with the application. These approaches assume that application developers create a template where they specify a set of service categories and their properties (i.e. functionalities) which are needed to achieve certain user tasks. The system chooses and allocates the appropriate services to match the specified task template.

On the other hand, interactive composition solutions assume a greater degree of user control and, as a result, provide their users with greater flexibility and freedom. The approaches that follow this path provide tools and user interfaces enabling users to compose applications [18–20]. The main difference between automated and interactive composition approaches lies in the fact that the first enables the system to act upon the users' needs and intents, while the second enables users to play an active role in defining the application functionalities they need. One specific subgroup of interactive composition approaches provides further flexibility by enabling end-user composition. These systems allow *'...end-user creativity to emerge in a ubiquitous environment where people can create their own niche applications or adapt their ubiquitous surroundings'* [21]. Usually, end-user composition approaches support user control via a visual editor - a tool utilized to create and edit composite applications and to visualize the environment in the user's vicinity. Some examples of editors can be found in Accord [10], OSCAR [12], Memodules [22] and PiP [23, 24].

Composition approaches in the ubicomp and ambient intelligence domains have been surveyed in several articles. Bronsted et al. [25, 26] categorized solutions and technologies according to service specifications, runtime and deployment characteristics, evaluation approaches and various qualitative trade-offs. Similarly, Urbeita et al. [27] compared existing approaches with regards to expressiveness of service descriptions, composition and execution models and execution environment. Bakhouya and Gaber [28] surveyed Web Service composition approaches and organized their categorization according to service-oriented architecture and Web Service generalities. Ibrahim and Le Moeul [29] focused on middleware solutions and classified the related work with respect to the main functionalities for a service composition middleware: interoperability, discoverability, adaptability, context-awareness, QoS management, security, and so on. Stavropoulos et al. [30] presented, perhaps, the most general survey which covered multiple aspects of composition approaches, including application domains, modelling of services, composition methods, knowledge representation and user interfaces. The work of Mavormaati et al. [31] presenting their conceptual framework for the design of IoT architectures supporting end-user development includes also a good survey on existing application composition platforms for IoT. In contrast to the existing surveys, this article focuses exclusively on interactive composition.

Our contribution is the following: We present a chronology of the most influential application composition platforms. We define the generic application composition process and categorize existing approaches accordingly. We describe the forms of user involvement in application composition approaches and analyze their prototypes for completeness. We also discuss and analyze user interfaces and various tools and technologies that enable users to compose and even control their application at runtime. Furthermore, based on related work, we identify three GUI metaphors: the pipeline, the jigsaw puzzle, and the join-the-dots metaphor. We argue that these three metaphors are repeatedly utilized in the interfaces for interactive application approaches.

A part of this overview article has been published in the first author's doctoral dissertation [32]. This article presents an extended and updated version of the overview with a deeper analysis of interactive composition approaches.

2 Summary of Application Composition Platforms

For this review we identified the most influential application composition platforms existing in the literature by scanning research articles from 2002 till 2015. Based on relevancy we have selected 25 of them. They are outlined chronologically in Table 1. The timing of each project was determined by project descriptions, publication dates of project deliverables and related articles available on the Internet. We contacted the authors of

Table 1. A brief chronology of influential composition platforms.

Platform	Reference	Timing	Composition
eGadgets	Mavrommati et al [21, 33]	2002–2004	Interactive, End-User
Aura	Sousa et al. [5, 17]	2002–2005	Task-Based
STEER	Masuoka et al. [13]	2002–2006	Task-Based
Accord	Rodden et al. [10]	2003–2005	Interactive
Compose Tool	Wisner & Kalofonos [34]	2004–2006	Task-Based
PiP	Chin et al. [23, 24]	2004–2006	Interactive
PalCom	Pollini et al. [7]	2005–2008	Interactive, End-User
Collaboration Bus	Gross & Marquardt [35]	2005–2009	Interactive
InterPlay	Messer et al. [11]	2006	Task-Based
HomeBird	Rantapuska & Lahteenmaki [36]	2006–2007	Task-Based
OSCAR	Newman et al. [12, 37]	2006–2008	Interactive
Deploy Spontaneously	Kawsar et al [18]	2006–2008	Interactive, End-User
Memodules	Mungellini [22]	2007	Interactive, End-User
ReWire	Vanderhulst et al. [19]	2007–2011	Interactive
Platform Composition	Pering et al. [38]	2007– 2011	Interactive, End-User
MAPPER	Rycerz et al. [39]	2007– 2013	Interactive, End-User
Touch & Compose	Sanchez et al. [15]	2008	Interactive
MEDUSA	Davidyuk et al. [40]	2009-2010	End-User
iCompose	Davidyuk et al. [9, 41]	2009-2011	Interactive
García-Herranz et al.	García-Herranz et al [42]	2009– ???	Interactive
SECREST	Grönvall et al. [8]	2010–2012	End-User
Preuveneers & Berbers	Preuveneers & Berbers [20]	2010–2012	Interactive
OntoCompo	Brel et al. [43]	2011–(ongoing)	Interactive, End-User
DroidCompo	Le [44]	2014	Interactive
Serna et al.	Serna et al. [45]	2014–(ongoing)	Interactive, End-User

the most recent publications to determine whether their project has an ongoing (or finished) status. Although the focus of our article is interactive application composition, we included several task-based solutions in this summary table as well. The reason is that some existing task-based systems are not fully automated and still need their users to be involved to some degree. This is necessary to compensate for the lack of influence of user goals or to simply supervise and control the process. Usually, users participate when defining their goals or when ranking (or choosing) from a variety of solutions provided by the composition system. Therefore, these task-based systems offer user interfaces for specifying users' goals and preferences and for choosing the final solutions. We survey the solutions with partial user involvement in this article as well.

3 The Generic Composition Process

The goal of any composition system is to produce an application or a composite service which matches the user's needs and preferences. Therefore, one of the most crucial characterizing aspects of any composition system is how the process of composing an application is organized and implemented. This section provides an overview of the generic composition process, as well as an examination of how these phases are supported by the analyzed platforms.

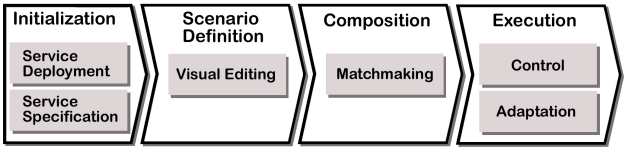

Fig. 1. Workflow of the generic composition process.

3.1 Composition Phases

Usually, an application is being composed in several consequent steps or phases which govern the life-cycle of applications. We analyzed existing interactive composition approaches and identified the most common phases which form the generic composition process namely: *initialization*, *scenario definition*,*composition* and *execution* (Figure 1). Some more advanced schemes could be applied, e.g. controlling the composition process based

on overall application performance [46]. Applications are also terminated at some point as well, but we do not consider termination in this article. Next, we will analyze each phase in more detail.

Initialization. In general, *initialization* implies creating the specifications of services and making the services available for application composition and usage. Approaches differ in *how* this phase is carried out. For example, some application composition solutions assume that, in addition to creating service specifications, services have to be deployed and configured by users [18, 21, 33]. Service deployment is also emphasized in the Memodules project [22] which uses the Lay&Play tool to create digital counterparts (i.e. avatars) of ubiquitous artifacts, and adds these counterparts to a repository, so that they become available for the other phases of the composition process. In contrast, interactive composition [19, 23, 24] and task-based computing solutions [5, 13] assume that the initialization phase is limited only to creating service specifications, and adding them to a repository that supports various ways of searching (i.e. querying) on these specifications.

Scenario definition. An application template or a scenario for the future application is created during this phase. Most automated composition approaches omit this phase, and assume that application templates are created during the initialization phase, although Sousa et al [5] and Messer et al [11] do envisage a phase during which predefined application templates are customized to match the user's needs. However, this phase is of important relevance in interactive (and especially end-user) application composition solutions. Systems using this approach provide editing tools based on a visual or a physical interface which captures the user's intent, and transforms it into a formal application template. The template is materialized during the next phase (*composition*) in a concrete application. This path is followed, for instance, by [12, 23, 24, 37]. On the other hand, some approaches use a different method which allows users to create and edit compositions made up of concrete appliances (i.e. ubiquitous devices). These editors perform two functions simultaneously: template creation and application composition [34].

Composition. This phase is central to the whole process, and usually, involves a composition mechanism (or an algorithm) which takes an application description as input and produces (i.e. composes) a concrete application using the service instances that match the functionality specified in the description. Composition mechanisms can be classified as automated [30, 32] or interactive. In this publication we will focus on the interactive one, in which the process of creating applications is governed by users through dedicated interface support [34, 36]. Some end-user composition approaches such as PalCom [7], CADEAU [41] and iCompose [9] do not use GUIs but physical user interfaces. We present a detailed analysis and discussion of user interfaces for interactive application composition in the next section of this article.

Execution. This phase governs application usage and various real-time aspects of application support, such as resource monitoring and adaptation control. Some solutions assume that the users themselves should be able to manually re-compose the application dynamically, as it is not entirely possible to proactively anticipate changes in a user's goals. This feature is supported, for example, in the CADEAU and iCompose prototypes [9, 41]. Automated monitoring and analysis of applications for diagnosing failures and supporting new services has been designed as part of the OSCAR project [37]. That project suggests sharing users' applications in a privacy-preserving manner, so that users are able to view and compare their composite solutions, and thus, choose the most appropriate ones that match their needs.

3.2 Analysis of Composition Completeness and Purpose

To investigate how these generic composition phases are supported by the systems in question, we introduce two aspects: completeness and purpose. In this article, under completeness we mean the degree the system developers have implemented all the steps which are necessary to compose one application. This aspect indicates whether the system in question has a work-in-progress status, or for example, the developers intentionally target some specific system functionality like composition mechanism, omitting application execution and control, for instance. The second characteristic, purpose, is linked to the nature of a composition system, its application domain, and reflects the kind of users the system is targeted for.

In this subsection, we analyze the completeness of existing composition solutions and how users are involved in each phase of their generic composition process. To that end, we classify the phases of solutions based on the degree of user involvement into manual, interactive, automated, and design-time. We consider the phase to be 'manual' if the system offers some user interface for the composition phase and the user has to define

Table 2. User involvement in different composition processes.

Solution	Initialization	Scenario Definition	Composition	Execution
eGadgets [21]	Manual	Manual	Manual	Interactive
Aura [5]	Design Time	Interactive	Automated	Automated
STEER [13]	Design Time	Interactive	Automated	Interactive
Accord [10]	Design Time	Manual	Manual	n/a
Compose Tool [34]	n/a	Manual	Automated	n/a
PiP [24]	n/a	Manual	Manual	Interactive
PalCom [7]	Manual	Manual	Manual	Manual
CollaborationBus [35]	Design Time	Manual	Manual	Manual
InterPlay [11]	Design Time	Interactive	Automated	Automated
HomeBird [36]	Design Time	Design-Time	Automated	Automated
OSCAR [12]	Design Time	Manual	Manual	Manual
Deploy Spontaneously [18]	Manual	Design Time	Interactive	Manual
Memodules [22]	Manual	Manual	Interactive	Interactive
ReWire [19]	Design Time	Interactive	Automated	Interactive
Platform Composition [47]	n/a	n/a	Manual	Automated
MAPPER [39]	Manual	End-User	End-User	Automated
Touch&Compose [15]	Design Time	n/a	Interactive	Automated
MEDUSA [40]	Design Time	Manual	Interactive	Interactive
iCompose [9]	Design Time	Design Time	Manual	Manual
García-Herranz et al [42]	n/a	Manual	Manual	Manual
SECREST [8]	n/a	Interactive	Manual	Automated
Preuveneers & Berbers [20]	n/a	Manual	Automated	Interactive
OntoCompo [43]	n/a	Manual	Manual	n/a
DroidCompo [44]	Design Time	Manual	Manual	Manual
Serna et al.[45]	n/a	Manual	Manual	Automated

all required details through this UI. On the other hand, we classify the phase as 'interactive' if the users are able to do the most of operations through some user interface but they receive support from the system. That is, the system controls or performs some part of the operation in an automated manner, although users may customize or configure the system's operations during this phase. In contrast, we classify the phase as 'automated' if the phase is entirely controlled by the system and users are not supposed to intervene at all. Finally, we attributed the phase as 'design time' if the content and operations during this phase are determined at the system design time. Similarly, we marked some phases as 'no answer' (n/a) for those systems that lack a description of how users are involved in this concrete phase.

Table 2 summarizes our findings. As can be seen, most of the solutions assume that initialization is performed at the design time. This is mainly due to the research focus and goals of the systems' creators. For example, the creators of ReWire [19] propose pervasive menus and interactive wizards to control the behavior of the pervasive system. The interactive wizards use the mixed-initiative interface approach to enable a dialog between users and the system when creating application scenarios (the authors call them 'tasks') and executing applications. However, initialization in ReWire is supported at the system design time and requires the intervention of application designers.

We consider Memodules [22], Deploy Spontaneously [18] and PalCom [7, 48, 49] to offer the most complete set of user tools across all the steps of the composition process. Memodules enables users to initialize application services from real objects (the authors refer to 'tangible objects' in their paper) by using Lay&Play tool. This tool is based on RFID technology and allows users to introduce application services to the system by taking pictures and registering RFID tags attached to the real objects. This tool creates a digital counterpart (i.e. an application service) and adds it to the list of available services in the system. Memodules Action Builder (see Figure 4) allows users to describe which services are involved and how they interact with each other in the concrete application scenario. Memodules features another tool, Memodules Console, for the applications composition and execution phases. Memodules Console is equipped with infrared sensors, and touch sensors, and RFID readers so that users can control application execution and also interact with their applications using the tool.

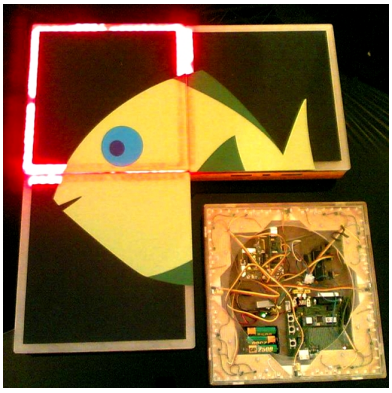

Fig. 2. Physical composition of tiles in the PalCom project [50]. Figure published under CC-Att license. Courtesy of Patrizia Marti and PalCom project.

Deploy Spontaneously employs the do-it-yourself approach to deploy, compose and execute smart home applications. This system assumes that services (the authors refer to 'ubiquitous artefacts' in their paper) are delivered to users as ready-to-run devices which are bundled with software. Users deploy services to the system through a tangible deployment tool which resembles Memodules Lay&Play. Each service is supplied with an RFID card which corresponds to an application and service profiles. However, Deploy Spontaneously supports only prepackaged applications, i.e. applications are developed by third-party developers and introduced into the system when users install the services associated to these applications. This is a limitation of the approach. Deploy Spontaneously allows users to control composition and execution of applications by manipulating the RFID cards provided with the services.

An interesting approach is used in the PalCom project [7, 48, 49]. This project enables doctors to create applications for treating disabled people during rehabilitation sessions in swimming pools. The PalCom project features a custom hardware platform consisting of a set of modular units or tiles. These tiles can communicate with other tiles, are able to recognize their relative positions, and can be composed into different patters, as shown in Figure 2. There are two kinds of tiles, master and ordinary ones. Master tiles are used for application configuration purposes (e.g. application initialization), while ordinary ones are used in rehabilitation sessions (i.e. during the application execution). The PalCom project combines the initialization and the scenario composition phases by providing a Migrating UI browser. This browser allows doctors to program rules (i.e. the behavior of tiles) and create the applications. Interesting is the fact that in PalCom the application composition and execution are performed simultaneously, as applications start executing while patients are composing tiles into patterns.

The generic end-user programming system for smart spaces utilizing rules (i.e. trigger-condition-actions structures) for composition process is suggested by Garcia-Herranz et al [42]. Although the authors omitted the initialization phase from the description of their system [42], the other phases rely on a set of visual editors and rule composers which support end-users with different expertise in IT. This system, perhaps, is the only thoroughly evaluated and tested approach among the ones reviewed in this article.

The e-Gadgets [21, 33], OSCAR [12, 37], Accord [10] and PiP [23, 24] and Serna et al. [45] projects focus on smart home applications. These five solutions assume that smart homes are populated with devices and consumer electronic appliances providing services that can be used as building blocks for applications. e-Gadgets provides a specific middleware, called GAS [33], that interfaces directly with the hardware and enables devices to communicate, share service descriptions, and interpret messages. GAS, like Deploy Spontaneously [18], also uses a do-it-yourself approach by supporting users in hardware deployment(i.e. smart home devices) and configuration with a special tool. On the other hand, OSCAR and Accord focus on the configuration and control aspects of smart home applications; hence, these two approaches assume that the initialization phase is performed by the application designers only. PiP, in contrast, focuses exclusively on the end-user functionality; thus, the initialization phase is not described in articles reporting this work. All five solutions, combine the scenario definition and application composition phases. This is achieved through visual editors which enable users to manipulate with services, i.e. create applications and control them dynamically. For this purpose, PiP offers the PiPView editor, in which users can define their applications by dragging and dropping service representations. PiP uses the UPnP protocol to discover the available services which are then shown to the users in the PiPView. On the other hand, e-Gadgets allows users to compose applications from services on the fly (adding services while the application is being executed) and configuring and adapting the existing applications as needed. In addition, both OSCAR and Serna et al. provide a user interface that allows users to select and control individual devices in a smart home based on sensed data. Neither Accord nor Serna et al. offer any functionality for the application execution phase, but

OSCAR, e-Gadgets and PiP allow the users to control the execution of their applications.

HomeBird [36], Compose Tool [34] and InterPlay [11] enable composition of smart home applications and apply the task-based approach. This approach assumes that users interact with the environment by specifying tasks that involve one or multiple devices. As mentioned in the Introduction, task-based solutions employ various automated mechanisms for application composition and execution; thus, users have limited control over their environment and applications. For example, in HomeBird tasks (i.e. applications) are created by the designers in the form of plugins, which are installed on users' mobile devices and are being executed in the background without user interaction. Each plugin requires one or multiple devices to be used simultaneously. The plugins are capable of searching dynamically for available devices in the user's proximity. As soon as the plugin discovers all the necessary devices, the HomeBird's UI presents the plugin to the user who may activate the plugin and interact with the application. As can be seen, the initialization and scenario definition phases are performed at the design time, while the composition and execution is done automatically. The users are involved only when the plugin is already composed. In contrast, Compose Tool [34] offers greater support for users by featuring a middleware and a visual editor (Compose Tool). This solution focuses exclusively on the scenario definition and composition phases; hence, the authors do not discuss the initialization and execution phases in their article. Compose Tool fuses the task-based approach with the end-user composition approach; thus, resulting in the solution in which the manual scenario definition phase is followed by the automated composition phase. The scenario definition is based on the editor through which users choose, according to their intuition, the devices to be involved in a task. When the devices have been selected, the system performs automated composition to find the possible combinations (i.e. tasks) between the chosen devices and proposes these tasks to the users. InterPlay [11] is another task-based system that involves users during the scenario definition phase. The initialization phase is performed at the system design time. The application composition and execution phases are fully automated. InterPlay offers a user interface through which users specify their tasks by issuing commands in pseudo English. Such a command consists of a verb (e.g. 'play') and a subject (e.g. 'movie') and an object (e.g. 'on a TV') which describe the user's task. The system interprets these pseudo sentences and automatically matches them with the available devices in the proximity of the user.

Two solutions, Aura [5, 17] and STEER [13] also employ the task-based approach to application composition. Aura's composition process features four phases. The users are involved in the scenario definition only while composition and execution phases are fully automated. STEER aims to present users with an abstract view on their services (and devices) which they can use on-the-fly. STEER, like Aura, assumes that users' services and devices are described and made available through the service discovery. Therefore, the initialization phase is performed at the design time. These two approaches implement the scenario definition phase differently. Aura offers a set of user interfaces through which users can provide the system with their QoS preferences and various trade-offs. Later, these preferences are taken into account during the composition and execution phases. Aura, unlike the other solutions, supports automated adaptation of the composed applications during their execution. That is, the authors consider several fault scenarios, in particular, changes in the users' QoS requirements, unavailability of services, and resource degradation. STEER enables users to create their tasks in a web browser-based editor which also allows users to control the execution of their tasks. Although STEER features an automated composition phase, the execution phase assumes some degree of user involvement. Indeed, users can edit, start, stop and schedule the execution of their tasks on-the-fly.

The Collaboration Bus [35] and the Platform Composition project [38, 47] provide two end-user solutions that focus on the application composition phase; hence, the other phases are not addressed in their work. Collaboration Bus relies on an editor which supports two kinds of users based on their technical experience: advanced users and novice users. Collaboration Bus and Platform Composition, similarly to other end-user solutions, combine the scenario definition and application composition phases. However, Collaboration Bus has a more sophisticated editor resembling a programming toolkit for developing applications, which allows to compose, control execution, and even test applications in a simulated environment. Platform Composition, in contrast, focuses only on the user interface for application composition. In addition, this solution does not offer any support for users during the application execution phase.

Touch&Compose [15], iCompose [9] and MEDUSA [40] assume that a smart space is populated with a number of services associated with RFID tags. Users compose applications by touching these tags with their mo-

bile terminals equipped with RFID readers. All these solutions utilize a combination of a physical interface with a traditional GUI. However, these systems concentrate on different phases of the composition process. That is, MEDUSA targets the scenario definition phase, Touch&Compose focuses on the composition phase, while iCompose supports exclusively the composition and execution phases. Services are created and introduced into the system at the design time. In addition, iCompose and Touch&Compose assume that application scenarios are developed by professional developers utilizing dedicated programming environments. MEDUSA, in contrast, offers end-users a tangible tool for scenario definition. Touch&Compose performs the application composition using automated mechanisms, while MEDUSA and iCompose offer a set of interfaces featuring support for end-users. In addition, these two approaches enable user control during the application execution phase.

The SECREST [8] prototype targets end-user application composition in the healthcare domain. The prototype's composition process consists exclusively of the scenario definition and composition phases that use the visual editor called 'SECREST Composer'. This editor serves two tasks: searching for available (i.e. currently executing) services and visual composition. The execution phase is realized automatically.

Preuveneers et al [20] target composition for smart home applications and propose a generic context-aware system which helps non-technical users to automate their homes. The authors assume that end users are able to install and connect new devices while the system detects them automatically (however, details of the initialization phase are omitted in their paper). Users create applications in a user interface which abstracts devices and their properties via a set of intelligent UI widgets. These widgets are linked to the composition tool using semantic descriptions representing all elements of the environment and various contexts. The execution of the applications is performed by the system, although the user is able to control interactively some of its run-time aspects.

OntoCompo [43] is targeted to software developers that want to create new applications upon existing ones. Authors called this approach Developer-Friendly Development (in contrast to End-User Development). OntoCompo's model describes each application in terms of links between tasks models, UI models and software models(i.e. assemblies of components). Developers compose new applications by combining UI elements from other different applications. Similary, DroidCompo [44] enables software developers to create simple service based applications (i.e. applications that delegate work to remote web services) for Android platform. In this case, the composition process is a little bit more complex than OntoCompo. DroidCompo provides models that allow developers to describe Android applications (including its UI), web services and web service clients. The composition engine utilize the three models to build native Android applications.

Finally, Rycerz et al.[39] presents a complex composition framework built for the MAPPER project. This framework is targeted to scientists who desire to combine software tools deployed in external research institutions. The environment presented in this paper "*supports ability to connect software modules to form large-scale, multiscale simulations and directly execute them on distributed e-infrastructures suitable for particular application models chosen by users*". Users themselves upload the metadata associated to the service they want to use into the system during the initialization phase.

4 Interactive Application Composition

In general, automated application composition approaches focus on minimizing the users' distraction in ubiquitous environments and they limit user involvement in the composition process (see Figure 1). However, an automated composition approach cannot guarantee that the resulting application will match the one anticipated by the users. Interactive application and end-user composition approaches address this issue. These approaches utilize visual(GUIs) and non-visual(physical user interfaces and AI-based mechanisms) tools to enable users themselves to compose, manage and control applications. Both interactive and end-user composition approaches actively advocate user involvement in all phases of the composition process. In this section, we overview the user interfaces and tools which are used in the scenario definition, composition and execution phases of these solutions.

4.1 User Interfaces and Tools for Scenario Definition and Composition

Interactive and end-user composition approaches involve the users in the application composition process by providing them tools to design and configure the ap-

plication. This section explores visual-based, as well as non-visual approaches suggested by the literature.

4.1.1 Visual Tools

Visual editors are the main instrument for interactive and end-user composition approaches to let users interact with the system during the scenario definition and composition phases. A composition editor visualizes service instances directly available for composition, provides an intuitive mechanism to create composite applications and, optionally, displays the state of the entire environment. Such a tool has a GUI which relies on a visual representation using graphics, animations or icons. A typical composition editor has at least two panels, a list of available service types or other elements (e.g. event templates) and an editing canvas (workspace). Users can drag and drop service icons or other visual elements into the editing canvas where these icons become links to concrete services in the environment. The workspace is an area where users arrange, connect and configure services.

The visual composition tools discussed in this section are closely related to visual programming languages. These languages rely on visual techniques throughout the programming process, and enable users to create, debug, and execute programs by manipulating icons or other visual representations [51].

The key element of any visual composition editor is the metaphor which gives the user cognitive hints and instantaneous knowledge about how to use this editor [52, 53]. Based on related work, we have identified three different metaphors: the pipeline, the jigsaw puzzle and the join-the-dots metaphor.

The pipeline metaphor. Applications are represented as directed graphs, where nodes correspond to single services, while links (i.e.,pipelines) are communication channels that connect two (or more) services together. Pipelines can be organized in complex structures using binary logic operators and can contain processing filter elements which trigger a predefined event when the data inside the template matches a specified template.

The pipeline metaphor is a powerful visual programming paradigm which makes it possible to model complex applications and their behavior. Despite its expressiveness, editors using this metaphor have steep learning curves. Therefore, it is either used in tools intended for expert users (e.g. CollaborationBus [35], MAPPER [39]), or it requires a high degree of automation if the editing tool is meant for non-expert users (e.g. Compose Tool prototype [34]). A more simplified version of this metaphor has been used in the SECREST [8] (Figure 3) and OSCAR [12] prototypes to support non-expert users. In SECREST, services support only single input/output interfaces, and a service can be a provider (i.e. yields data), a consumer (i.e inputs data) or both. SECREST features a work-pane that overviews available services and a workspace where users can freely arrange and connect services to make applications. OSCAR, on the other hand, allows users to operate only with pipeline templates. A pipeline template is an application consisting of two 'slots', a source and a destination, that both have to be assigned with a service or some content. The editor uses a wizard-like approach to guide users through the steps that are necessary to populate a template.

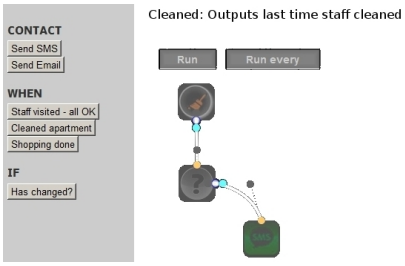

Fig. 3. The GUI of SECREST Composer which relies on a simplified pipeline metaphor [8]. Courtesy of E.Grönvall.

The jigsaw puzzle metaphor. By far the most commonly used metaphor. A jigsaw puzzle-based editor models each available service as a piece of a puzzle. This metaphor promises an easy to understand interface in which non-expert users can easily associate each piece of a puzzle with the visual service it represents [54]. Moreover, the shapes of the pieces of the puzzle provide the cognitive clues necessary to understand the possible actions. This metaphor permits more complex configurations than, for instance, the pipeline metaphor [35]. The puzzle metaphor supports complex structures (e.g., clusters of pieces of the same type) that act as single entities. Clusters facilitate end-user application design by decreasing complexity. In addition, the puzzle metaphor supports collaborative multi-user application composition, as two or more users can work on the same application simultaneously. The downside of the jigsaw puzzle approach stems from the fact that the metaphor has constrained potential for expression: the pieces of the puzzle can have a limited number of interfaces (i.e. sides) thus narrowing the range of possible application scenarios.

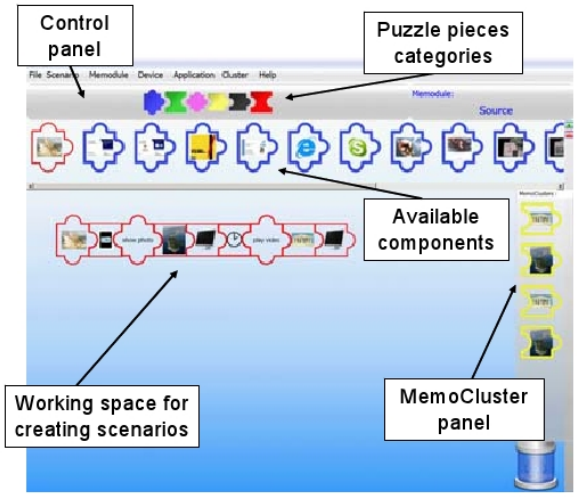

Fig. 4. The editors' interfaces of the Memodules prototype [22] which is based on the jigsaw puzzle metaphor. ©2007 ACM, Inc. http://dl.acm.org/citation.cfm?doid=1226969.1227016 . Published with permissions.

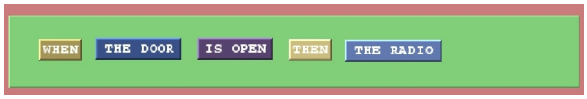

Fig. 5. A part of the 'magnetic poetry' composition GUI presented by García-Herranz et al [42] which relies on the tile-based approach. Courtesy of Pablo Haya.

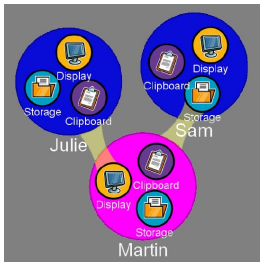

Fig. 6. The editor of the Platform Composition prototype [38, 47, 59] which is based on the join-the-dots metaphor. ©2009 Springer. With kind permission from Springer Science+Business Media:[38]

This metaphor has been used for instance in the Accord prototype [10, 55] to specify the interconnection of simple sensors, web applications and various multimedia devices and services. The Action Builder editor (Figure 4) developed in the Memodules project [22] uses a similar approach to compose event-based applications. In this case, events are modelled as pieces of a puzzle and can be composed together with certain services.

The recent success of open visual programming languages that use the jigsaw metaphor (e.g. Google Blockly [56] and Scratch [57] has favoured that some researchers adopt those languages for their platforms. This is the case of Serna et al. [45] that uses Blockly to build their platform to control smart devices at home.

One variation of this metaphor is the tile-based approach [58] which is a generalization of the jigsaw puzzle metaphor. This approach allows users to construct programs by manipulating and combining graphical building units (called tiles). Although each tile can be combined only with certain other tiles, the shapes of tiles are not physically limited, and can be connected with an arbitrary number of other tiles. For example, the tile-based approach used in application composition is presented by Garcia-Herranz [42]. In their system, applications are represented using abstract entities that are governed by executing a set of rules. The authors proposed the 'magnetic poetry' interface (see Figure 5) in which users organize and manage their rules and entities.

The join-the-dots metaphor. In this case, the editing canvas presents a set of individual devices that are available in the environment. Each device is shown as the center of a cluster, while the surrounding nodes represent services accessible from the environment. Users create compositions linking one service to the desired destination device. In particular, this metaphor has been applied in the editor of the Platform Composition prototype [38, 47, 59] (Figure 6). A different version is presented by Mavrommati and Kameas [21]. Their model, called the Plug-Synapse model, represents individual devices as circular nodes, and allows connections to be created by drawing lines between them. However, the plug-synapse model does not exhibit services offered by individual devices, thus making visual representation more compact. When drawing a line (i.e., making a composition) between two devices, users specify the composition on an 'association matrix' that visualizes a selection of concrete capabilities (i.e., services) offered by the devices chosen.

The main advantages of this metaphor resides in the simplicity of its visual representation. Only the services and devices available for composition are visualized. In addition, the state of the entire environment is shown in a single screen. Indeed, it also shows the devices and services that are not used in any composition, which at a certain point may become a disadvantage as it only permits environments with a few devices.

Other metaphors. A user interface for visual composition does not necessarily need to utilize a GUI metaphor. For example, Preuveneers [20] suggested a

set of intelligent UI widgets which allow users to create applications and control them. Each widget presents to end-users an aspect of the system's functionality (e.g. available devices, preferences, etc) in a concise and uniform manner. These widgets are automatically mapped to the low-level configuration values and instructions for devices and services. To achieve this, the system uses principles of context-awareness with ontology- and rule-based inference which allow to translate users' high-level goals (captured via intelligent UI widgets) into low-level device compositions. In other words, these widgets offer a level of abstraction thus hiding the complexity of the environment and its configuration from users.

Also, we place here a few automated application composition approaches that adhere to a task-computing principle, namely STEER [13], InterPlay [11], Sousa et al.'s prototype [5] and Wisner and Kalofonos' solution [34]. These solutions feature a fully automated composition phase and involve users during the initialization and scenario definition phases. While strictly speaking these are not interactive application composition solutions, they nevertheless provide visual tools for users during the initialization and scenario definition phases.

Often, these approaches suggest drop-down menus, where each menu is associated with a service, to initialize the concrete application, like in STEER [13]. When all the services have been configured, users trigger the automatic composition phase from the task management screen. Another approach suggests so called composition screens, where users are able to specify the connections of services and devices for concrete application, like in the InterPlay prototype [11]. InterPlay relies on visual 'verb-object-target' constructions which resemble pseudo-English sentences. Users specify a task by choosing first a 'verb' (i.e., a command), then a 'object' (i.e., content) and, finally, a 'target' (i.e., a target device). Although this approach clearly offers a natural and an easy-to-learn user interface, users are not really able to compose tasks, but rather trigger the automated composition of the tasks that were introduced into the system at the design time. Another example utilizing composition screen is suggested by Sousa et al. [5], Figure 7a. Similarly to InterPlay [11], this system suffers from the fact that only pre-programmed tasks are supported. However, Sousa's composition screen, in addition to scenario definition, supports the initialization phase also. That is, the screen allows users to specify their task-related QoS preferences, or, particularly, a trade-off between different QoS dimensions (e.g. latency and quality of content), before they trigger the automated composition, Figure 7b. Although default QoS dimension preferences are provided for each task by the task designers, users are able to override default values using the composition screen.

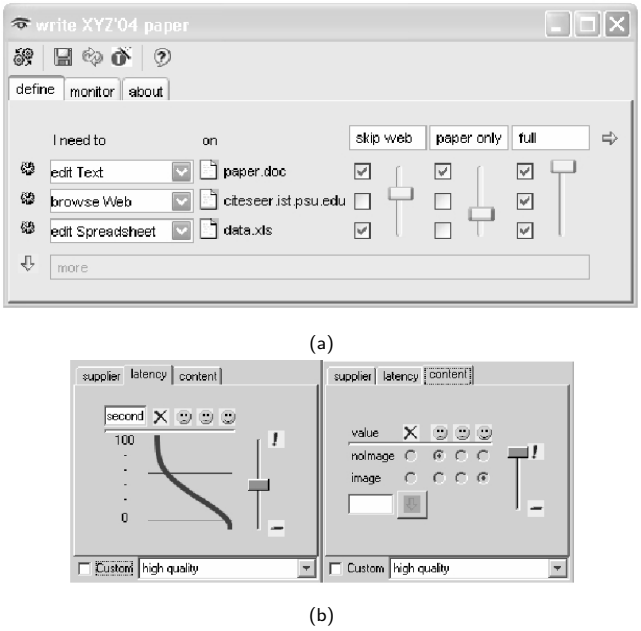

Fig. 7. The task composition interface (a) and the interface for capturing QoS preferences (b) designed by Sousa et al. [5, 17]. ©2006 IEEE. Reprinted, with permission, from [5].

In contrast, Wisner and Kalofonos [34] (Figure 8) combine the scenario definition and composition phases. Their Compose Tool allows users to select services for composition (interactive scenario definition), while the system instantly suggests possible interconnections between these services (automated composition). These phases are performed in the Compose Tool editor simultaneously, as follows: an editor's GUI consists of three panes, where the left pane shows a list of all known ser-

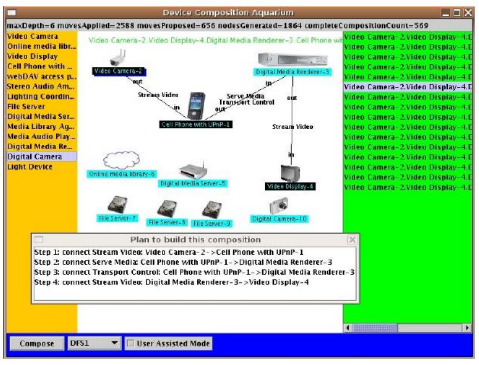

Fig. 8. The composition editor presented by Wisner and Kalofonos [34]. ©2007 IEEE. Reprinted, with permission, from [34]

vice types, the middle pane is an editing canvas which shows the services available in the environment, and the right pane shows the compositions proposed by the system. When users drag-and-drop different service types to the editing canvas, the composition phase is immediately triggered. The system associates the service types through the service discovery with the service instances available in the environment and proposes the interconnection of these instances. Then, users can choose from multiple proposals the one that best suits their needs. Similarly, OntoCompo [43] also combines scenario definition and composition phases. In this case, application developers can manipulate and combine GUI elements from multiple applications in order to compose a new one.

4.1.2 Non-visual Tools

In addition to visual composition editors based on GUI metaphors, some researchers advocate using non-visual composition tools, such as programming by example [60], and physical user interfaces [61]. These tools are used in different phases of the composition process.

The Pervasive interactive Programming (PiP) prototype [23, 24] presents the programming by example concept. PiP combines a visual editor (used in the scenario definition and the composition phases) and an AI mechanism that supports users during the composition phase. Users select the devices they want to combine together in the visual editor. After that, and during the composition phase, the users interact with the real devices like they would interact with the device in the execution phase. The AI mechanism observes this interaction and detects patterns that are transformed into an application. This method has also been used in the Active Surfaces prototype [49] (part fo the PalCom project). This prototype permits the physical connection of multiple autonomous tiles with each other (Figure 2). Individual tiles can recognize their position. Expert users (therapists in this case) create physical assemblies of tiles which follow a certain pattern during the scenario definition phase. After that, therapists program the tiles by showing the correct pattern of tiles to the system. Later, during the composition phase, tiles have to be placed in the correct pattern by end users (i.e., by patients) when they take part in rehabilitation sessions.

This method's advantage lies in its ability to hide most of the details of the underlying mechanisms from users. This fact makes this approach useful for non expert users. However, as pointed out by Mavrommati et al [62], *'there are cases when there cannot be a task example performed - i.e. because the application splits between different locations, different time periods or in situations that cannot be replicated.'*

Non-visual composition tools can also be built using physical user interfaces which rely on physical objects rather than on traditional I/O devices [61]. For example, MEDUSA prototype [40] uses the jigsaw puzzle metaphor during the scenario definition phase. In this case, instead of visual representations in a screen, the puzzle pieces are RFID enabled carton pieces presenting different icons. The icons can be arranged in various two-dimensional structures (sequences) which form a composite application as needed by users. An RFID enabled mobile phone is used to capture these structures by touching the icons sequentially. In contrast, the iCompose [9] and Touch&Compose prototypes [15] utilize the physical interface during the scenario definition and the composition phase which are performed simultaneously. In this case, the physical user interface consists of RFID tags distributed in the environment and advertising the services available locally. Each tag is associated to a single device or service. Users inform the system about the services and devices they want to include in the application by touching the tags with the RFID enabled device.

4.2 User Interfaces and Tools for Application Execution

Execution phase of composed application may need user control at runtime, because various kinds of user input are required to deal with non-trivial situations. For example, Vanderhulst et al [19] and Hardian et al [63] argue that it would be unrealistic to delegate all decision-making to the system and expect it to figure out by itself what the user wants. In this case, user input is needed to deal with changes in user goals or context. For instance, users might need to manually specify services (or devices) to be used with the application while it is being executed. This need is addressed in the iCompose and Touch&Compose prototypes [9, 15] which rely on a physical interface. This interface allows users to manually adapt their applications at runtime by touching the required devices in their local environment. In contrast, Vanderhulst et al [19] proposed a graphical user interface which is generated at runtime and evolves with the environment. When a user triggers a reconfiguration of an application, the GUI loads the interactive

wizard which provides access to application-specific features of the currently active application. Most of the analyzed systems anticipate the users' need for application management during the execution phase; hence, they provide basic support for starting and stopping composed applications, e.g. like in CollaborationBus [35], PiP [23, 24], OSCAR [12], eGadgets [21], ReWire [19] and the prototypes of Garcia-Herranz et al [42] and Preuveneers et al [20]. A few approaches provide the application control using a physical interface. For example, the Memodules prototype [22] employs a tangible console for controlling multimedia applications during the execution phase. This works as follows: To begin with, users touch the console with various augmented objects in order to identify and trigger the corresponding application. After the console retrieves and starts the application, the users can control (i.e., interact with) the multimedia application using various touch sensors to play the next audio recording, the previous one, to stop, and to adjust the volume. A similar idea was proposed in the Deploy Spontaneously prototype [18] where the authors used RFID cards to control RFID-enabled augmented objects at runtime. In their prototype, each augmented object is supplied with a set of RFID cards, and an integrated RFID reader. Each card corresponds to a certain command (install/uninstall a service, start/stop application execution). Users control augmented objects and application execution by simply swiping a card - this action identifies the command (associated with the card) and the augmented object (associated with the RFID reader).

4.3 Analysis

We classified related work according to the user interfaces and user tools for the scenario definition, composition and execution phases. We analyzed whether the approaches use a visual metaphor (e.g., join-the-dots, jigsaw puzzle, pipeline or other) or a non-visual tool (e.g., a tangible UI). The solutions that do not support user involvement during the scenario definition, composition or execution phases are not discussed further. The summary of results is shown in Table 3. It should be noted that the scenario definition and the composition phases are combined in this summary table reflecting the fact that most (though not all) interactive composition solutions combine these two phases in their composition process.

To sum up, non-visual tools are effective for only relatively simple applications and special cases when, for example, users need to compose applications by choosing physical devices. Non-visual tools tend to be used in a combination with an automated composition mechanism or a visual tool, as has been demonstrated in related work [12, 21].

In contrast, visual tools for application composition support a much wider range of applications and user groups. However, the suitability of a concrete visual editor for an application or a user group is heavily influenced by the metaphor used. The pipeline metaphor is a powerful and flexible visual tool which enables the composition of complex applications, albeit at the expense of steeper learning curve. Thus, this metaphor can be recommended for complex applications and expert users. The jigsaw puzzle metaphor, in contrast, offers a natural visual interface, and is thus easy to understand. However, the supported applications have rather simple structures. This metaphor is therefore more appropriate for simple applications and inexperienced users. The join-the-dots approach offers an easy to understand representation and application structures that go beyond trivial cases. We recommend this metaphor to inexperienced users, especially when they need to be able to comprehend the state of the entire environment.

User control in runtime application composition raises a number of open challenges which have not been fully addressed in the related work. One reason could be that system designers assume that user activities and needs are fixed and do not change after an application has been composed. This is correct in situations where the composed applications have a relatively short lifespan, and focus only on users' immediate goals. On the other hand, recent research tends to promote continuous user interaction in ubiquitous environments [64], which results in a prolonged lifespan of applications, and gives rise to additional challenges related to application adaptivity. Hence, users need to have the possibility to control and adapt applications to accommodate changes in their goals and preferences at runtime. This requires developing various user interfaces (e.g., tangible, speech and gesture-based) to be used during the application execution phase in order to provide a better user experience.

5 Conclusions and Future Work

In this article, we give an overview of interactive application composition approaches that focus on supporting the user's activities and needs in ubiquitous envi-

Table 3. User tools for the scenario definition, composition and execution phases in different solutions.

Solution	Reference	Scenario Definition, Composition	Execution
eGadgets	[21]	Visual: join-the-dots	GUI
Aura	[5, 17]	Visual: other metaphors	-
STEER	[13]	Visual: other metaphors	GUI
Accord	[10]	Visual: jigsaw puzzle	-
Compose Tool	[34]	Visual: pipeline	-
PiP	[24]	Mixed: GUI & Prog. by example	GUI
PalCom	[7]	Non-visual: Physical UI	Physical UI
CollaborationBus	[35]	Visual: pipeline	GUI
InterPlay	[11]	Visual: other metaphors	-
HomeBird	[36]	-	GUI
OSCAR	[12]	Visual: pipeline	GUI
Deploy Spontaneously	[18]	Tangible UI	Tangible UI
Memodules	[22]	Visual: jigsaw puzzle	GUI & Tangible UI
ReWire	[19]	-	GUI
Platform Composition	[47]	Visual: join-the-dots	-
MAPPER	[39]	Visual: pipeline	-
Touch&Compose	[15]	Tangible UI	Tangible UI
MEDUSA	[40]	Mixed: Tangible UI & jigsaw puzzle	GUI & Tangible UI
iCompose	[9]	Tangible UI	Tangible UI
García-Herranz et al	[42]	Visual: jigsaw puzzle	GUI
SECREST	[8]	Visual: pipeline	-
Preuveneers & Berbers	[20]	Visual: widget-based	GUI
DroidCompo	[44]	Visual: other metaphors	GUI
OntoCompo	[43]	Visual: other metaphors	GUI
Serna et al.	[45]	Visual: jigsaw puzzle	-

ronments. We classify the existing approaches into automated, interactive and end-user categories according to the degree of user involvement and control. The solutions that fall into the automated category assume that user involvement in application configuration, management and provisioning tasks is kept to a minimum. These solutions often rely on task-based principles and feature a fully-automated composition phase in their composition process. We surveyed several automated solutions that offer user tools for the initialization and scenario definition phases. In contrast, interactive solutions provide users with various user interfaces and tools to enable users to compose, manage and control applications throughout their composition process. End-user composition is considered as a special case of interactive application composition that features profound user involvement in the scenario definition and composition phases. In this approach, these two phases are often integrated into one single step.

In addition, in this article we presented a general process for application composition consisting of four phases. We gave a phase by phase analysis of how the existing solutions implemented and organized their composition processes. For each solution we identified whether it targeted some specific system's functionality (i.e., application specification) and focused on some specific kind of users (i.e., novice and inexperienced users).

Furthermore, we gave an overview of user interfaces and visual tools for application composition. We classified the user interfaces into visual and non-visual tools. We classified further the visual tools according to the GUI metaphor used in their interfaces. We presented the most commonly used GUI metaphors: the pipeline, the jigsaw puzzle, and the join-the-dots metaphor. We discussed the pros and cons of each metaphor and presented how they are implemented in existing visual editors. With this overview, our aim is to provide means for describing and classifying different composition approaches, and for comparing these approaches.

One of the future challenges for application composition, yet to be resolved, is related to the generalizability of interactive composition approaches. Currently, generic solutions such as solutions using task-based computing principles (or end-user composition solutions for home environments) are lacking in supporting various user experience levels. That is, the functionality and the user interfaces offered by these solutions are developed in order to meet the needs of some ab-

stract 'average' user. Even worse, most reviewed solutions do not make efforts either to study needs of their potential users or portrait their experience and skills. Since a common standard specifying this does not exist yet, the range of functionality and the complexity of user interfaces are motivated entirely by the imagination of their developers. As a result, only a few solutions can be truly tested and used outside research labs where they have been developed. However, this is not applicable to domain specific solutions which only focus on the users with some previously known level of experience (for example, museum curators in the Cicero Designer prototype [65]). Fortunately, with the emerge of simple end-user visual programming languages such a Scratch [57] and Google Blockly [56] the complexity of user interfaces for composing tools can be significantly reduced. Some of the most recent composing tools (e.g. the one presented by Serna et al. [45]) are integrating these visual programming languages in their tools.

Another interesting issue is related to GUI metaphors and how they are applied in the related work. The reviewed approaches use only one metaphor at most. This is acceptable when composite applications are relatively simple and the variety of possible application scenarios is restricted, for example, due to few services' types being available in the environment, like in the Memodules project [22]. On the contrary, if a composition approach aims to facilitate users to compose complex application scenarios, we expect that a combination of various metaphors is required at the same time. GUI metaphors can be compared to various views into the same application: changing metaphors allows highlighting the application's details which otherwise are not easy for users to understand. In addition, it is necessary to keep in mind that if the same metaphor is understood by expert users, for example, it does not necessary mean that it will be similarly comprehended by inexperienced users. However, it is challenging to design a user interface which combines several GUI metaphors simultaneously. The initial step in this direction has been made in the PiP prototype [23] where a GUI metaphor was successfully combined with the programming by example technique.

Given the strong involvement of users in ubiquitous environments, service composition approaches have to design proper interfaces to interact with users. An important issue related to the user interface concerns finding the balance between system autonomy and user control in interactive composition approaches. For example, a fully autonomic system is delegated the maximum autonomy in all decisions made. In opposite, a fully user controlled system delegates all the decision-making to the user and the system executes the user's commands only. Managing the tradeoff between both orientations depends on particular features of ubiquitous environments (e.g., characteristics of users and the kind of services provided by the environment). The challenge is then to design a user interface that allows to adjust the tradeoff between user control and system autonomy with respect to specific features of ubiquitous environments.

Our last but not the least remark stems from the fact that interactive composition approaches support user activities and needs which belong to a much larger context of users' everyday lives. Therefore, it is important to study and understand what motivates users to rely on composition approaches and how they match users' habits and fit users' routines. This is also an essential step to allow composition technologies to be deeper integrated into the background of users' lives while making them truly disappearing and pervasive as was envisioned by Mark Weiser [2]. We believe that the future work on interactive composition, while bringing new approaches and tools, will also observe users' routines and study how these new approaches complement users' lives in greater detail.

References

[1] D. McIlroy. Mass-Produced Software Components. In P. Naur and B. Randell, editors, *Proceedings of Software Engineering Concepts and Techniques*, pages 138–155, Garmisch, Germany, 1969. NATO Science Committee, Scientific Affairs Division.

[2] M. Weiser. The Computer for the 21st Century. *Scientific American*, 265(3):94–104, September 1991.

[3] M. Schuster, A. Domene, R. Vaidya, S. Arbanowski, M. Kim, W. Lee, and H. Lim. Virtual Device Composition. In B. Werner, editor, *Proceedings of the 8th International Symposium on Autonomous Decentralized Systems (ISADS'07)*, pages 270–278, Washington, DC, USA, March 21-23 2007. IEEE Computer Society.

[4] E. Mugellini, O. Abou Khaled, S. Pierroz, S. Carrino, and H. Chabbi Drissi. Generic Framework for Transforming Everyday Objects into Interactive Surfaces. In J. A. Jacko, editor, *Proceedings of the 13th International Conference on Human-Computer Interaction, Part III (HCI'09)*, LNCS 5612, pages 473–482, San Diego, CA, July 19-24 2009. Springer.

[5] J. Sousa, V. Poladian, D. Garlan, B. Schmerl, and M. Shaw. Task-based Adaptation for Ubiquitous Computing. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, 36(3):328–340, May 2006.

[6] J. M. Paluska, H. Pham, U. Saif, G. Chau, C. Terman, and S. Ward. Structured Decomposition of Adaptive Applications. *Pervasive and Mobile Computing*, 4(6):791–806, December 2008.

[7] A. Pollini, E. Grönvall, P. Marti, and A. Rullo. Constructing Assemblies in the Health Care Domain: Two Case Studies. In L. Chittaro et al., editors, *Proceedings of the Italian Workshop of Mobile Guide (as part of Virtuality Conference 2006)*, Torino, Italy, July 2006.

[8] E. Grönvall, M. Ingstrup, M. Pløger, and M. Rasmussen. REST based Service Composition: Exemplified in a Care Network Scenario. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pages 251–252, Sept. 2011.

[9] O. Davidyuk, E. Gilman, I. Sánchez, J. Mäkipelto, M. Pyykkönen, and J. Riekki. iCompose: Context-Aware Physical User Interface for Application Composition. *Central European Journal of Computer Science*, 1(4):442–465, 2011.

[10] T. Rodden, A. Crabtree, T. Hemmings, B. Koleva, J. Humble, K.-P. Akesson, and P. Hansson. Between the Dazzle of a New Building and Its Eventual Corpse: Assembling the Ubiquitous Home. In *Proceedings of the 5th Conference on Designing Interactive Systems (DIS'04)*, pages 71–80, Cambridge, MA, USA, 2004. ACM.

[11] A. Messer, A. Kunjithapatham, M. Sheshagiri, H. Song, P. Kumar, P. Nguyen, and K. H. Yi. InterPlay: a Middleware for Seamless Device Integration and Task Orchestration in a Networked Home. In *Proceedings of the 4th Annual IEEE Conference on Pervasive Computing and Communications (PERCOM'06)*, pages 296–307, Pisa, Italy, March 2006. IEEE Computer Society.

[12] M. Newman, A. Elliott, and T. Smith. Providing an Integrated User Experience of Networked Media, Devices, and Services through End-User Composition. In J. Indulska et al., editors, *Proceedings of the 6th International Conference on Pervasive Computing (Pervasive'08)*, LNCS 5013, pages 213–227, Sydney, Australia, 2008. Springer.

[13] R. Masuoka, B. Parsia, and Y. Labrou. Task Computing - the Semantic Web meets Pervasive Computing. In K. Sycara and J. Mylopoulos, editors, *Proceedings of the 2nd International Semantic Web Conference (ISWC'03)*, LNCS 2870, pages 866–881, Sanibel Island, Florida, USA, Oct 2003. Springer.

[14] S. Ben Mokhtar, N. Georgantas, and V. Issarny. COCOA: COnversation-based Service Composition in PervAsive Computing Environments with QoS Support. *Journal of Systems and Software*, 80(12):1941–1955, 2007.

[15] I. Sánchez, J. Riekki, and M. Pyykkönen. Touch&Compose: Physical User Interface for Application Composition in Smart Environments. In J. Langer, editor, *Proceedings of the International Workshop on Near Field Communication*, pages 61–66, Upper Austria, 2009. IEEE Computer Society.

[16] C. C. Vo, T. Torabi, and S. Loke. *Task-Oriented Systems for Interaction with Ubiquitous Computing Environments*, volume 73 of *LNCS 8197*, pages 332–339. Springer Berlin Heidelberg, 2012.

[17] J. P. Sousa, B. Schmerl, P. Steenkiste, and D. Garlan. *Activity-Oriented Computing*, chapter 186, pages 3215–3241. Software Applications: Concepts, Methodologies, Tools, and Applications. IGI Global, Hershey, USA, 2009.

[18] F. Kawsar, T. Nakajima, and K. Fujinami. Deploy Spontaneously: Supporting End-Users in Building and Enhancing a Smart Home. In H. Y. Youn and W.-D. Cho, editors, *Proceedings of the 10th International Conference on Ubiquitous Computing (UbiComp'08)*, pages 282–291, NY, USA, 2008. ACM.

[19] G. Vanderhulst, K. Luyten, and K. Coninx. ReWiRe: Designing Reactive Systems for Pervasive Environments. In N. Graham and P. Palanque, editors, *Proceedings of the 15th International Workshop on Interactive Systems, Design, Specification, and Verification (DSV-IS'08)*, LNCS 5136, pages 155–160, Kingston, Canada, July 2008. Springer.

[20] D. Preuveneers and Y. Berbers. Intelligent Widgets for Intuitive Interaction and Coordination in Smart Home Environments. In *Proceedings of the 8th International Conference on Intelligent Environments (IE'2012),*, pages 157–164, june 2012.

[21] I. Mavrommati, A. Kameas, and P. Markopoulos. An Editing Tool that Manages Device Associations in an In-Home Environment. *Personal Ubiquitous Computing*, 8(3-4):255–263, 2004.

[22] E. Mugellini, E. Rubegni, S. Gerardi, and O. A. Khaled. Using Personal Objects as Tangible Interfaces for Memory Recollection and Sharing. In B. Ullmer and A. Schmidt, editors, *Proceedings of the 1st International Conference on Tangible and Embedded Interaction (TEI'07)*, pages 231–238, Baton Rouge, Louisiana, 2007. ACM.

[23] J. Chin, V. Callaghan, and G. Clarke. An End-User Tool for Customising Personal Spaces in Ubiquitous Computing Environments. In H. Jin, T. Laurence T., and J.-P. Jeffrey, editors, *Proceedings of the 3rd International Conference on Ubiquitous Intelligence and Computing (UIC'06)*, LNCS 4159, pages 1080–1089, Wuhan, China, 2006. Springer.

[24] J. Chin, V. Callaghan, and G. Clarke. An End-User Programming Paradigm for Pervasive Computing Applications. In *Proceedings of the ACS/IEEE International Conference on Pervasive Services (ICPS'06)*, pages 325–328, Lyon, France, June 2006. IEEE Computer Society.

[25] J. Brønsted, K. M. Hansen, and M. Ingstrup. Service Composition Issues in Pervasive Computing. *IEEE Pervasive Computing*, 9(1):62–70, 2010.

[26] J. Brønsted, K. M. Hansen, and M. Ingstrup. Survey of Service Composition Mechanisms in Ubiquitous Computing. In *Proceedings of Workshop on Requirements and Solutions for Pervasive Software Infrastructures*, 2007.

[27] A. Urbieta, G. Barrutieta, J. Parra, and A. Uribarren. A Survey of Dynamic Service Composition Approaches for Ambient Systems. In *Proceedings on Ambi-Sys Workshop on Software Organisation and Monitoring of Ambient Systems (SOMITAS'08)*, pages 1–8, Quebec City, Canada, 2008. ICST.

[28] M. Bakhouya and J. Gaber. *Service Composition Approaches for Ubiquitous and Pervasive Computing Environments: A Survey*, pages 323–350. Information Science Reference/IGI Publishing, 2008.

[29] N. Ibrahim and F. Le Mouel. A Survey on Service Composition Middleware in Pervasive Environments. *International Journal of Computer Science Issues*, 1:1–12, August 2009.

[30] T. Stavropoulos, D. Vrakas, and I. Vlahavas. A Survey of Service Composition in Ambient Intelligence Environments. *Artificial Intelligence Review*, pages 1–24, 2011. 10.1007/s10462-011-9283-1.

[31] I. Mavrommati, G. Birbilis, and J. Darzentas. A conceptual framework for the design of IoT architectures that support end-user development. *Networking Science*, 3(1-4):71–81, dec 2013.

[32] O. Davidyuk. *Automated and Interactive Composition of Ubiquitous Applications*. PhD thesis, University of Oulu, Oulu, Finland, June 2012.

[33] I. Mavrommati and A. Kameas. End-User Programming Tools in Ubiquitous Computing Applications. In C. Stephanidis, editor, *Proceedings of the 10th International Conference on Human-Computer Interaction (HCI'03)*, pages 172–177, Crete, Greece, June 2003. Lawrence Erlbaum Associates.

[34] P. Wisner and D. Kalofonos. A Framework for End-User Programming of Smart Homes Using Mobile Devices. In Y. Jiang, editor, *Proceedings of the 4th IEEE Consumer Communications and Networking Conference (CCNC'07)*, pages 716–721, Las Vegas, NV, USA, 2007. IEEE Computer Society.

[35] T. Gross and N. Marquardt. Creating, Editing and Sharing Complex Ubiquitous Computing Environment Configurations with CollaborationBus. *Scientific International Journal for Parallel and Distributed Computing*, 11(3):289–303, September 2010. URI: http://www.scpe.org/index.php/scpe/article/view/661/. Cited 2012/04/23.

[36] O. Rantapuska and M. Lahteenmaki. Task-based User Experience for Home Networks and Smart Spaces. In A. Zaslavsky and K. Truong, editors, *Proceedings of the International Workshop on Pervasive Mobile Interaction Devices*, pages 188–191, Sydney, Australia, 2008.

[37] M. Newman and M. Ackerman. Pervasive Help @ Home: Connecting People Who Connect Devices. In *Proceedings of the International Workshop on Pervasive Computing at Home (PC@Home)*, pages 28–36, Sydney, Australia, 2008.

[38] T. Pering, R. Want, B. Rosario, S. Sud, and K. Lyons. Enabling Pervasive Collaboration with Platform Composition. In H. Tokuda et al., editors, *Proceedings of the 7th International Conference on Pervasive Computing (Pervasive'09)*, LNCS 5538, pages 184–201, Nara, Japan, May 2009. Springer.

[39] K. Rycerz, M. Bubak, E. Ciepiela, D. Harężlak, T. Gubała, J. Meizner, M. Pawlik, and B. Wilk. Composing, execution and sharing of multiscale applications. *Future Generation Computer Systems*, 53:77–87, dec 2015.

[40] O. Davidyuk, N. Georgantas, V. Issarny, and J. Riekki. *MEDUSA: A Middleware for End-User Composition of Ubiquitous Applications*, pages 197–219. Handbook of Research on Ambient Intelligence and Smart Environments: Trends and Perspectives. IGI Global, August 2011.

[41] O. Davidyuk, I. Sánchez, and J. Riekki. *CADEAU: Supporting Autonomic and User-Controlled Application Composition in Ubiquitous Environments*, chapter 4, pages 74–103. Pervasive Computing and Communications Design and Deployment: Technologies, Trends, and Applications. IGI Global, 2011.

[42] M. García-Herranz, P. Haya, and X. Alamán. Towards a Ubiquitous End-User Programming System for Smart Spaces. *Journal of Universal Computer Science*, 16(12):1633–1649, june 2010. |http://www.jucs.org/jucs_16_12/towards_a_ubiquitous_end|.

[43] C. Brel, P. R. Gonin, A. Giboin, M. Riveill, and A.-M. Dery. Reusing and Combining UI, Task and Software Component Models to Compose New Applications. In *Proceedings of the 28th International BCS Human Computer Interaction Conference on HCI 2014*, pages 1–10. BCS, sep 2014.

[44] C. Le. A model driven framework for modeling and composing service based Android applications. In *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, pages 450–457, New York, New York, USA, mar 2014. ACM Press.

[45] M. A. Serna, C. J. Sreenan, and S. Fedor. A visual programming framework for wireless sensor networks in smart home applications. In *2015 IEEE Tenth International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pages 1–6, 2015.

[46] E. Gilman, I. Sánchez, M. Cortés, and J. Riekki. Towards User Support in Ubiquitous Learning Systems. *Learning Technologies, IEEE Transactions on*, 8(1):55–68, 2015.

[47] T. Pering, K. Lyons, R. Want, M. Murphy-Hoye, M. Baloga, P. Noll, J. Branc, and N. De Benoist. What Do You Bring to the Table?: Investigations of a Collaborative Workspace. In *Proceedings of the 12th ACM International Conference on Ubiquitous Computing (Ubicomp '10)*, pages 183–192, New York, NY, USA, 2010. ACM.

[48] E. Grönvall, L. Piccini, A. Pollini, A. Rullo, and G. Andreoni. Assemblies of Heterogeneous Technologies at the Neonatal Intensive Care Unit. In *Proceedings of the European Conference on Ambient Intelligence (AmI'07)*, LNCS 4794, pages 340–357. Springer, 2007. DOI: 10.1007/978-3-540-76652-0_21.

[49] E. Grönvall, P. Marti, A. Pollini, and A. Rullo. Active Surfaces: a Novel Concept for End-User Composition. In A. Mørch et al., editors, *Proceedings of the 4th Nordic Conference on Human-Computer Interaction (NordiCHI'06)*, pages 96–104, Oslo, Norway, 2006. ACM.

[50] E. Grönvall, P. Marti, A. Pollini, and A. Rullo. Active Surfaces: A Novel Concept for End-user Composition. In *Proceedings of the 4th Nordic Conference on Human-computer Interaction: Changing Roles*, pages 96–104, 2006.

[51] M. Boshernitsan and M. Downes. Visual Programming Languages: A Survey. Technical Report UCB/CSD-04-1368, University of California, Berkeley, CA, USA, 2004. URI: http://nma.berkeley.edu/ark:/28722/bk0005s5d5t. Cited 2012/04/23.

[52] T. D. Erickson. Working with Interface Metaphors . *The Art of Human–Computer Interface Design*, pages 65–73, 1990.

[53] A. Marx. Using Metaphor Effectively in User Interface Design. In *CHI'94 Conference Companion on Human Factors in Computing Systems*, pages 379–380, New York, NY, USA, 1994. ACM.

[54] E. Kandogan. JigsawTree: Design of a Task Composition Interface for Complex Applications. In *Proceedings of the 8th IFIP International Conference on Human-Computer Interaction (INTERACT'01)*, pages 561–568, Tokyo, Japan, July 2001. IOPress.

[55] J. Humble, A. Crabtree, T. Hemmings, K.-P. Akesson, B. Koleva, T. Rodden, and P. Hansson. Playing With Your Bits: User Composition of Ubiquitous Domestic Environments. In A. Dey, A. Schmidt, and J. McCarthy, editors, *Proceedings of the 5th International Conference on Ubiquitous Computing (UbiComp'03)*, LNCS 2864, pages 256–263, Seattle, WA, USA, October 2003. Springer.

[56] Google. Blockly.

[57] MIT. Scratch.

[58] L. Cavallaro, E. D. Nitto, C. A. Furia, and M. Pradella. A Tile-Based Approach for Self-Assembling Service Compositions. In R. Calinescu, editor, *Proceedings of the 15th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'10)*, pages 43–52, Oxford, UK, 2010. IEEE Computer Society.

[59] R. Want, T. Pering, S. Sud, and B. Rosario. Dynamic Composable Computing. In M. Spasojevic and M. Corner, editors, *Proceedings of the 9th Workshop on Mobile Computing Systems and Applications (HotMobile'08)*, pages 17–21, Napa Valley, California, February 2008. ACM.

[60] A. Dey. End-User Programming: Empowering Individuals to Take Control of their Environments. In D. Olsen and S. Klemmer, editors, *Proceedings of the CHI 2005 Workshop on the Future of User Interface Design Tools*, Portland, Oregon, USA, 2005.

[61] J. Riekki. RFID and Smart Spaces. *International Journal of Internet Protocol Technology*, 2(3-4):143–152, 2007.

[62] I. Mavrommati and J. Darzentas. End-User Development in AmI: a User Centered Design Overview of Issues and Con-

cepts. *e-Minds: International Journal on Human-Computer Interaction*, 1(3):87–104, December 2007.

[63] B. Hardian, J. Indulska, and K. Henricksen. Exposing Contextual Information for Balancing Software Autonomy and User Control in Context-Aware Systems. In A. Zaslavsky and K. Truong, editors, *Proceedings of the Workshop on Context-Aware Pervasive Communities: Infrastructures, Services and Applications*, pages 253–260, Sydney, Australia, 2008.

[64] S. Balandin and H. Waris. Key Properties in the Development of Smart Spaces. In C. Stephanidis, editor, *In Proceedings of the 5th International Conference on Universal Access in Human-Computer Interaction (as part of HCI'09)*, LNCS 5615, pages 3–12, San Diego, CA, USA, July 2009. Springer.

[65] G. Ghiani, F. Paternò, and L. D. Spano. Cicero Designer: An Environment for End-User Development of Multi-Device Museum Guides. In *Proceedings of the 2nd International Symposium on End-User Development (IS-EUD'09)*, LNCS 5435, pages 265–274, Siegen, Germany, 2009. Springer.