Opportunistic software composition: benefits and requirements

Charles Triboulot, Sylvie Trouilhet, Jean-Paul Arcangeli, Fabrice Robert

Opportunistic software composition: benefits and requirements

Charles Triboulot[12], Sylvie Trouilhet[23], Jean-Paul Arcangeli[2] and Fabrice Robert[1]
[1] *Sogeti High Tech, Aeropark 3 Ch. Laporte 31300 Toulouse, France*
[2] *Université de Toulouse, UPS, IRIT, 118 route de Narbonne 31062 Toulouse, France*
[3] *IUT A, Av. G. Pompidou CS 20258 81104 Castres, France*
Firstname.Lastname@irit.fr, Firstname.Lastname@sogeti.com

Keywords:
Opportunism, software component, automatic bottom-up composition, ambient systems

Abstract:
Traditional software development relies on building and assembling pieces of software in order to satisfy explicit requirements. Component-based software engineering simplifies composition and reuse, but software adaptation to the environment remains a challenge. Opportunistic composition is a new approach for building and re-building software in open and dynamic contexts. It is based on the ability to compose software components in a bottom-up manner, merely because they are available at a point and not because the construction of a specific software has been demanded. In this way, software emerges from the environment. This paper analyzes the advantages of such an approach in terms of flexibility and reuse, along with the requirements that an infrastructure supporting opportunistic composition should satisfy: it should be decentralized, autonomic, and dynamically adaptive. The state of the art of automatic software composition shows that few solutions are actually bottom-up, and that none of them fully satisfies the requirements of opportunistic composition.

1 Introduction

When rationalizing software development, two major points should be considered: productivity and reuse on the one hand, capability of evolution on the other hand. Regarding these points, component-based software engineering improves object-based software engineering, mainly due to the explicitation of the required interfaces at the same level as the provided ones.

In traditional software engineering, composition is triggered when a need is made explicit and when the building of the software is demanded. Components are identified, then selected among existing ones or developed one by one, in order to compose the application. This approach is called top-down because the development is driven by the satisfaction of explicit requirements. In a general way, meeting requirements is a constant challenge for software engineers.

Once developed, applications are deployed in social and technical contexts that are changing. Therefore requirements change, thus applications must be flexible enough and adapt. But, even if pieces of solution exist, developers generally struggle to guess possible evolutions and to design points of variability in software. Because stakeholders' requirements are considered as a foundation at the early steps of traditional software engineering and architectural design processes, considering changes remains a major challenge in software engineering and software architecture.

Composition and adaptation issues are particularly crucial for ambient, ubiquitous and mobile systems. Given the large number of networked devices that populate ambient environments, as each of them may host software components, there are possibly numerous applications and services that can be built and provided to people in order to enrich their environment. Ambient intelligence can be measured in the yielded additional value while limiting the active participation of users. Ambient systems, with their dynamics, must be proactive and intelligent enough

to adapt according to the context in order to react to some situations, anticipate user needs, or go further than the predefined behavior.

1.1 Opportunistic composition

Our work aims to explore a new approach for the engineering of component-based software applications. Our approach, called "opportunistic composition", differs from the traditional one because composition is no longer driven by explicit requirements but opportunity-driven. Opportunistic composition is the ability to assemble software components merely because they are available in the surrounding environment. In this way, composition is directed by the execution context and by the opportunity to assemble components that are ready for composition (but which have not been developed specifically for the application under construction), and no longer by the necessity of satisfying explicit requirements. Thus, applications emerge from the environment and can evolve afterwards by dynamic (re)composition based on new opportunities.

Opportunism in software construction is a new idea, which has been little, if at all, studied. A paper on this subject has been presented at the French conference UBIMOB (Ubiquity and Mobility) in 2011 (Vergoni et al., 2011). In (Conti and Kumar, 2010), opportunistic computing has been defined as a new computing paradigm derived from *ad hoc* and opportunistic networking tightly coupled with social networking, which aims at exploiting available resources in an environment to provide distributed collaborative computing services and applications; the main challenges that are stated concern networking: intermittent connectivity, delay tolerance, protocol heterogeneity.

Of course, opportunistic composition raises several issues. The main ones relate to what can be composed (heterogeneity, composability...), what should be composed and how combinatorial complexity can be controled, context-awareness, the semantics of the resulting application, its quality attributes (security, reliablility, efficiency...), its viability and usefulness, and how to automatically perform opportunistic composition.

1.2 Contents and plan

This paper presents an analysis of the opportunistic composition approach. It discusses its advantages in terms of flexibility and reuse, and focuses on the feasibility and on the requirements that an infrastructure should satisfy in order to support opportunistic composition and allow the emergence of component-based software systems. Finally, the state of the art of automatic composition is analyzed in relation to these requirements. However, the paper does not present any architecture or experimental results. The contribution sets in the analysis of the opportunistic approach, in the identification of the requirements of its realization, and in the analysis of the state of the art.

The paper is organized as follows. Section 2 presents a real-world illustrating scenario which is useful for highlighting different situations of composition. In Section 3, opportunistic composition is defined more formally and the requirements for its realization are exposed. Section 4 analyzes the state of the art of automatic composition. To conclude, in Section 5, we give some directions for the realization of opportunistic composition.

2 Scenario

This section presents a scenario in which opportunistic composition assists users in their life and their job, and provides applications depending on the context. Here is Plip, an engineer who works in a secured experiment room of her company. She uses a robotic arm and a laser to project rays on a flat surface.

The scenario consists of four acts and shows several situations of opportunistic composition. It supports an analysis presented in Section 3. It underlines how an opportunistic composition system can react to some situations when they occur in a dynamic environment. Only the system's behavior is described, not its implementation. It is worth pointing out that compositions become possible without the necessity to explicitly specify user needs. They are realized merely because the components are available in the environment and because assembling them may be relevant. Plip does have requirements, but not necessarily explicited, and nevertheless the system provides an answer to them in a proactive way.

To carry out this scenario, we assume that the components are interoperable. Issues, such as security or privacy, are set aside in this work.

Act I - Plip is working in her office, on a new experimental model stored on her PC. Once

the model is completed, she runs an application on her tablet computer that provides estimated results according to the model. This application detects that a component implements Plip's model, thanks to a Wi-Fi connection between Plip's PC and the tablet, and proposes the use of this component. Since it matches her requirement, she accepts. Then, the application runs.

Act II - Since the estimated results are satisfactory, Plip is ready to perform the genuine experiment and compare the estimated results to the real ones. She goes to the experiment room with her tablet. The experiment room has been secured recently: it is isolated from Wi-Fi signals, but Plip does not know that. She enters the room with her tablet which is still connected by Wi-Fi with her PC. As the Wi-Fi component of the tablet is no longer working, a new composition is done, which involves a Bluetooth component which radio waves cross the walls of the secured room. This automatic composition preserves the communication link between the tablet and the outside PC. Thus, the application is still running and connectivity issues are imperceptible for Plip.

Act III - Plip starts the experiment. She turns on the different devices, oscilloscopes and screens, in the experimental room. She also turns on the central PC to control the experiment. The different devices are assembled automatically in order to propose an adequate experimental environment (see Figure 1). This is possible because the opportunistic composition system has learned, from previous experiments, the context in which such a configuration is relevant. Furthermore, one of the oscilloscope in the room has been upgraded, and now proposes a greater quality of service than the other oscilloscopes. Consequently, it is used in the assembly.

Act IV - While the experiment is running, the central PC suddenly breaks down. Unfortunately, it embeds several indispensable components for the experimental assembly, and plays an essential role in the command of the robotic arm ("Exp Leader" component). The PC is also used to collect the results of the experiment, and then analyze them. To allow the experiment to go on without waiting for repairs, the opportunistic composition system searches for other suitable components. In this case, the system may have several choices and must efficiently select and perform the most relevant ones (for instance an older "Exp Leader" component and a "Stock" component embedded in another device), with the help of context information.

3 Analysis of the approach

From this scenario, we extract three main advantages of the opportunistic approach compared to top-down and traditional approaches: proactiveness, flexibility and genericity. Then, we enumerate the functional and extrafunctional requirements which seem to be fundamental for an opportunistic composition system.

3.1 Benefits

Proactiveness - While Plip carries on her daily tasks, she never expresses explicitly her needs. However, the system proposes and maintains useful applications according to the current situation without relying on explicit user needs (explicit request of a component by the user is possible but not mandatory).

The final assembly is characterized as emergent, because neither the developer, the user, the system nor the components have a knowledge of it before its bottom-up fabrication. The key is to select useful compositions with methods related to learning or context awareness. The development of a proactive system, which does not demand requirement formalization and proposes relevant compositions, is promising in an ambient scope: in such undetermined and unpredictable environments, it is hard to express, or even identify, what the user needs can be.

Flexibility - Several adaptive reactions of the system can be observed in the scenario. Thanks to opportunistic composition, applications using appropriate components may replace ineffective and useless ones. The adaptive reactions are not explicitly expressed, but are the result of bottom-up decisions. On the one hand, this approach supports openness (components may appear or disappear from the local environment). On the other hand, applications are resilient (they are continually maintained and enhanced). Here, initial composition and resilience are supported by the same opportunistic approach.

Genericity - No hypothesis is made on the presence of certain components, their business domain, their type or number. Most of all, the system can work without the knowledge of the user

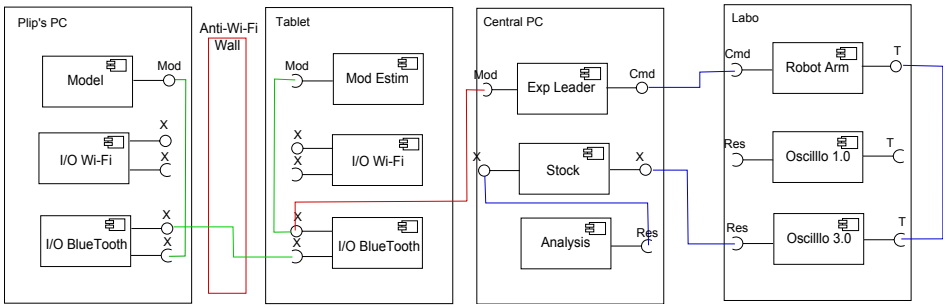

Figure 1: Component view of Act III

needs. So, an opportunistic composition system can be generic and useful in any (dynamic) environment. The system assisting Plip could also be used in various contexts (e.g. smart-home, guidance, monitoring...).

However, these benefits are guaranteed only if several open problems are resolved. How is the relevance of a composition precisely evaluated? How is the usability of an emergent application guaranteed without formalization of needs? How is the semantics of the application presented to the user? How is the combinatorial complexity mastered? The next sections will identify requirements for the realization of opportunistic composition. Meeting these requirements should bring answers to those questions.

3.2 Functional requirements

From the different composition situations underlined in the scenario, five functional requirements can be extracted. In the following enumeration, each of them is presented through the event that triggers the new composition.

1) Request of a Component (fRC) (Act I). A component is requested by a user or the system itself, for being a part of an application. Thus, this component tries to compose itself as a priority and properly fulfill its functionality. This is a way to implicitly express a user need without formalizing it.

2) Context Evolution (fCE) (Act III and Act IV). Some external situations may trigger a system reaction. Context awareness allows the system to detect these situations, while learning capabilities allow to memorize them. So, the system can properly react to environment evolution with the proper responses.

3) Appearance of a Component (fAC) (Act III). Due to the high dynamics of the environment, one or several components may be discovered by other components. This event can follow the start-up of a device or a mobility situation. The appearing component can be included in an existing assembly or trigger a new composition.

4) Disappearance of a Component (fDC) (Act II and Act IV). A component may disappear from the environment, in case of mobility, obsolescence or failure. The system should detect this situation and maintain the applications which were using the missing component by proposing new efficient compositions. Note that the disappearance of a component may be progressive and can be detected and anticipated before it actually happens, especially during a situation of mobility.

5) Upgrade (fU)(Act III). Some intrinsic properties of components may also dynamically evolve. For example, quality of service or interface profiles may be modified at runtime, further to an upgrade. These modifications are most likely to trigger new compositions. An opportunistic composition system should be able to identify any of these situations and react accordingly. It must also take into account that they can occur in any quantity and in any order. Previous situations are therefore considered as functional requirements for an opportunistic composition system.

3.3 Extrafunctional requirements

The aim of this section is to identify the extrafunctional requirements for the design of a system able to perform opportunistic composition. They represent generic issues the opportunistic system has to deal with, but do not make any assumptions on the methods and/or technologies used.

Seven requirements can be extracted and formalized. They can be used for the evaluation of different composition systems.

1) Decentralization (efDc): Ability to propose a decentralized process in which each task is attributed to an autonomous entity in a synchronous or asynchronous way. Indeed, the sys-

tem must work at a local level and consider the local neighborhood of components but not the whole environment. Furthermore, a local view is the natural way to perform bottom-up composition and obtain emergent results. The relevance of decentralized design is confirmed by the inability for a central entity to handle high dynamics, bottlenecks and/or failures.

2) Dynamic Adaptation (efDA): Ability to handle undetermined and unpredictable environments. The opportunistic composition system runs in highly dynamic and unpredictable environment: components may appear or disappear at runtime, even context and users' actions can change. Thus, it should be open and able to generic and efficient adaptation.

3) Combinatorial Optimization (efCO): Ability to handle a fair amount of possible compositions among available components. Ignoring this point may lead to efficiency and speed issues for the composition process. Thus, the system must handle this problem and efficiently select the most useful compositions using discriminating strategies.

4) Recomposition (efRc): Ability to dynamically maintain and/or enhance existing compositions. In order to support the resilience of applications, components may be added or replaced, or the entire assembly may be challenged.

5) Learning and Context awareness (efLC): Ability to take into account past activities and context to perform relevant compositions.

6) Utility of the Result (efUR): Ability to guarantee a useful application as a result of the composition. The emergent result must be useful and satisfy implicit needs. Its utility can be automatically verified, in local or global scope, *a posteriori* by feedback on the assembly, or *a priori* by identification of promising compositions.

7) Silence of User Needs (efSUN): Ability of the system to operate without the preliminary expression of user needs. This requirement is a main one in order to let the system propose emergent solutions without explicit requirements or user needs.

Next section confronts the state of the art with the functional and extrafunctional requirements.

4 State of the Art

We analyzed eight research works on component automatic composition in order to determine if and how they could answer the requirements of the opportunistic approach (Vallée et al., 2005; Grondin et al., 2006; Desnos et al., 2007; Bartelt et al., 2008; Rouvoy et al., 2009; Guidec et al., 2010; Sykes et al., 2011; Bonjean et al., 2013). Table 1 indicates both if the solution could handle the functional requirements (if the system could detect and then properly handle the situation, the corresponding box contains a cross) and the extrafunctional requirements (evaluated from - - to ++).

None of the solutions has (x) or (++) for all the requirements. The fDC (Disappearance of a Component) requirement is globally met. This is due to the systematic self-adaptation aspects developed in the reviewed works. Likewise, some extrafunctional requirements, such as efDc (Decentralization) or efDA (Dynamic Adaptation), are admitted as essential for automatic composition and a lot of solutions exists. However, efCO (Combinatorial Optimization) and efLC (Learning Context), although they are extremely important in dynamic environments, have yet to be considered. A similar assessment can be made for functional requirements; situations such as fU (Upgrade) and fCE (Context Evolution) also seem to be hardly anticipated. The last issue is the status of the expression of user needs, many systems demanding their formalization before making any composition decision. This expression is not mandatory, but it can be noted that efUR (Utility of the Result) and efSUN (Silence of User Needs) are not satisfied together (see Table 1).

5 Conclusion and perspectives

In this paper, we have defined opportunism as an approach for building and re-building software by composing available software components. We have illustrated the advantages of this approach: flexibility and adaptiveness. In our opinion, opportunistic software composition is not limited to ambient systems: for example, it could assist software engineers, which have to compose reusable components as part of large libraries or to adapt component-based software, by proposing them relevant compositions.

Opportunistic composition is a promising generic approach, well-adapted to highly dynamic and open environments, which disregards formalization of user needs. However, the link between needs and component assemblies should be con-

	fRC	fCE	fAC	fDC	fU	efDc	efDA	efCO	efRc	efLC	efUR	efSUN
Vallée 2005		x		x		+	++	+	-	+	+	- -
Grondin 2006		x	x	x		- -	+	-	++	+	++	- -
Desnos 2007				x	x	- -	+	+	++	- -	-	+
Bartelt 2008	x			x		++	-	-	- -	- -	-	+
Rouvoy 2009	x			x	x	-	++	- -	+	-	+	-
Guidec 2010	x		x	x		++	++	-	- -	-	++	- -
Sykes 2011		x	x	x		+	+	-	++	-	-	-
Bonjean 2013	x	x	x	x	x	++	+	++	+	+	+	-

Table 1: Functional and extra-functional requirements

sidered through learning, evaluation of utility and user's feedback, in order to control the emergence. Thus, opportunistic composition may be a basis for design of intelligent systems, which proactively provide emergent applications adapted to (possibly unforeseen) situations, and anticipate user needs.

Systems that support opportunistic software composition should meet several requirements, in particular those related to combinatorial complexity and relevance of the emergent applications. In order to face these challenges, we currently develop a solution based on multi-agent systems (MAS) which offer several advantages from an architectural point of view (Arcangeli et al., 2014), and precisely on cooperative MAS (Georgé et al., 2011), which support the emergence of functions through local interactions between cooperative agents. In order to select relevant compositions, agents have learning capabilities and are able to consider user's feedback.

References

Arcangeli, J.-P., Noel, V., and Migeon, F. (2014). Software Architectures and Multi-Agent Systems. In Oussalah, M. C., editor, *Software Architectures*, volume 2, chapter 5, pages 171–208. Wiley-ISTE.

Bartelt, C., Fischer, B., and Rausch, A. (2008). Towards a Decentralized Middleware for Composition of Resource- Limited Components to Realize Distributed Applications. In *3rd Int. Conf. on Pervasive and Embedded Computing and Communication Systems (PECCS 2013)*, pages 245–251.

Bonjean, N., Gleizes, M.-P., Maurel, C., and Migeon, F. (2013). Forward Self-combined Method Fragments. In *Agent-Oriented Software Engineering XIII*, number 7852 in LNCS, pages 168–178. Springer.

Conti, M. and Kumar, M. (2010). Opportunities in opportunistic computing. *Computer*, 43(1):42–50.

Desnos, N., Huchard, M., Urtado, C., and Vauttier, S. (2007). Automated and Unanticipated Flexible Component Substitution. In *Proc. of 10th Int. Symp. on Component-Based Software Engineering*.

Georgé, J.-P., Gleizes, M.-P., and Camps, V. (2011). Cooperation. In Di Marzo Serugendo, G., Gleizes, M.-P., and Karageogos, A., editors, *Self-organising Software*, Natural Computing Series, pages 7–32. Springer.

Grondin, G., Bouraqadi, N., and Vercouter, L. (2006). MaDcAr: An Abstract Model for Dynamic and Automatic (Re-)Assembling of Component-Based Applications. In *Component-Based Software Engineering*, number 4063 in LNCS, pages 360–367. Springer-Verlag.

Guidec, F., Sommer, N. L., and Mahéo, Y. (2010). Opportunistic Software Deployment in Disconnected Mobile Ad Hoc Networks. *Int. Journal of Handheld Computing Research*, 1:24–42.

Rouvoy, R., Barone, P., Ding, Y., Eliassen, F., Hallsteinsen, S., Lorenzo, J., Mamelli, A., and Scholz, U. (2009). Music: Middleware support for self-adaptation in ubiquitous and service-oriented environments. In *Software Engineering for Self-Adaptive Systems*, pages 164–182. Springer-Verlag.

Sykes, D., Magee, J., and Kramer, J. (2011). FlashMob: Distributed Adaptive Self-Assembly. In *Proc. of the 6th Int. Symp. on Software Engineering for Adaptive and Self-Managing Systems*, pages 100–109.

Vallée, M., Ramparany, F., and Vercouter, L. (2005). A Multi-Agent System for Dynamic Service Composition in Ambient Intelligence Environments. In *The 3rd International Conference on Pervasive Computing (PERVASIVE 2005)*, pages 175–182.

Vergoni, C., Tigli, J.-Y., Rey, G., and Lavirotte, S. (2011). Construction Bottom-up d'applications ambiantes en environnements partiellement connus a priori. In *7èmes Journées Francophones Mobilité et Ubiquité (UBIMOB 2011)*.