Composition and Compositional Reasoning: A Key to Tackling Complexity

Awais Rashid
Computing Department, Infolab21, Lancaster University, Lancaster LA1 4WA, UK
marash@comp.lancs.ac.uk

The advent of the modern information age has led to the daily routine of working with computers and employing a wide range of software systems to access and share information. In order to service the increasing needs of the information society, software is becoming large, complex and distributed. This is particularly true of ambient computing applications where software is ubiquitous and integrated with the day-to-day working practices of the people. Software servicing such ambient computing needs must be adaptive to its context and the needs of its users.

As the complexity of software increases, the **separation of concerns principle** [1] – which enables developers to focus on and reason about one specific concern in isolation from other concerns – becomes essential. Consequently, the software industry has adopted a range of separation of concerns techniques (and supporting methods, tools and environments) including components, objects, patterns, architecture styles and middleware approaches to manage software complexity and improve quality attributes.

Most separation of concerns techniques facilitate reasoning about concerns in isolation through abstraction and modularisation – this is also referred to as **modular reasoning** [2]. Composition of the isolated concerns is also often extensively dealt with, e.g., for object-oriented and component-based development. However, mostly, modular reasoning capabilities of such techniques tend to be the motivating factor for their adoption. More recently, techniques such as aspect-oriented software development, model-driven development and feature-oriented programming have emerged. All these techniques have one unifying characteristic: they all focus on improving the separation of concerns in a system. However, even more significantly, composition takes on a key role in these approaches. These techniques are attracting interest not only due to their inherent support for modular reasoning about concerns pertaining to a software system but also **compositional reasoning**, i.e. reasoning about the dependencies and interactions of the concerns [3]. This, in turn, allows one to reason about the global and emergent properties of the composed software, which is particularly useful when engineering ambient computing environments.

If we are to tackle the complexity of software systems, be it intrinsic complexity of the problem itself or extrinsic complexity of the various software technologies used to solve that problem, composition and compositional reasoning need to take centre stage. Only then can we hope to address problems of building large-scale, complex, distributed and ubiquitous software systems.

References

