End User Architecting

Vishal Dwivedi

School of Computer Science, Carnegie Mellon University
5000 Forbes Avenue, PA, 15213, USA
vdwivedi@cs.cmu.edu

Abstract. A large number of domains today require end users to compose various heterogeneous computational entities to perform their professional activities. However, writing such end user compositions is hard and error prone. My research explores an improved approach for design, analysis and execution of such end user compositions. I propose a new technique called 'end user architecting' that associates end user specifications in a particular domain as instances of architectural styles. This allows cross-domain analyses, systematic support for reuse and adaptation, powerful auxiliary services (e.g., mismatch repair), and support for execution, testing, and debugging. To allow a wider adoption of this technique, we have designed a framework that can be instantiated across a large number of domains, with composition models varying from dataflows, pub-sub, and workflows. This approach can reduce the cost of development of end user composition platforms (compared to developing them from scratch) and improve the quality of end user compositions.

1 Research Problem

Within an increasing number of domains an important emerging need is the ability for technically naive users to compose computational elements into novel configurations. Examples include e-science (e.g., astronomers who create new analysis pipelines to process telescopic data), intelligence analysis (e.g., policy planners who process diverse sources of unstructured text to discover socio-technical trends), and medicine (e.g., researchers who process repositories of brain imaging data to discover new disease pathways). In these domains professionals typically have access to a large number of existing applications and data sets, which must be composed in novel ways to gain insight, carry out "what if" experiments, generate reports and research findings, etc.

Unfortunately, assembling such elements into coherent compositions is a non-trivial matter [1][2]. In particular, we can identify five critical barriers.

1. **Excessive technical detail:** Existing languages and tools require end users to have knowledge of a myriad of low-level technical detail such as parameters, low-level control flow decisions, exception handling, and other programming constructs.
2. **Inappropriate computational models:** The computational models provided by typical execution platforms, such as SOA, may require end users to map their tasks into a computational vocabulary that is quite different from the natural way of decomposing the task in that domain. For example, tasks that are logically represented in the end user's mind as a workflow may have to be translated into the very-different vocabulary of service orchestrations and execution scripts.
3. **Inability to analyze compositions:** There may be many restrictions on legal ways to combine elements, dictated by things like format compatibility, domain-specific

processing requirements, ordering constraints, and access rights to data and applications. Today, discovering whether a composition satisfies these restrictions is largely a matter of trial and error, since there are few tools to automate such checks. Moreover, even when a composition does satisfy the composition constraints, its extra-functional properties — or quality attributes — may be uncertain. For example, determining how long a given computation will take to produce results on a given data set can often be determined only by time-consuming experimentation.

4. **Lack of support for reuse:** An important requirement in many communities is the ability for professionals to share their compositions with others in those communities. For instance, brain researchers may want to replicate the analyses of others, or adapt an existing analysis to a different setting (e.g., execute on different data sets). Packaging such compositions in a reusable and adaptable form is difficult, given the low-level nature of encodings, and the brittleness of the specifications.
5. **Impoverished support for execution.** Compared to the capabilities of modern programming environments, end users have relatively few tools for things like compilation into efficient deployments, interactive testing and debugging (e.g., setting breakpoints, monitoring intermediate results, etc.), history tracking, and graceful handling of run-time errors. This follows in part from the fact that in many cases compositions are executed in a distributed environment using middleware that is not geared towards interactive use and exploration by technically naive users.

This gap between the needs of end users and todays technology has two adverse consequences: (i) *The cost of producing effective compositions is high* because end users must become experts in implementation details not relevant to their primary task. (ii) *The quality is low* because compositions tend to be brittle and in many cases fail to meet their extra-functional requirements. Also, compositions are difficult to reuse, modify, and maintain, leading to gratuitous reinvention and errors.

2 Research Approach

My approach to address this problem has been to view end user composition activity as engaging in a high-level architectural design within a domain-specific style and to represent those end user architectures explicitly. Figure 1 illustrates the overall organization of end user composition tools and the placement of an architectural layer.

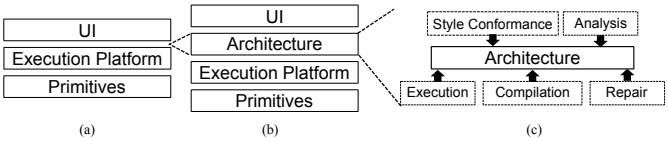

Fig. 1: End-user Architecting Approach

Part (a) of the figure shows the current state of affairs: users must translate their tasks into the computational model of the execution platform, and become familiar with the low-level details of that platform and primitive computational elements (applications, services, files, etc.) — leading to problems outlined in Section 1. Part (b) illustrates the new approach. Here, end-user architectures are explicitly represented as architectural models defined in a domain-specific architectural style. These models and the supporting infrastructure can then support a host of auxiliary services, including checking for

style conformance, quality attribute analysis, compilation into efficient deployments, execution and debugging mechanisms, and automated repair — as shown in part (c).

By decomposing the problem in this way we identify a new field of concern, which we term *end-user architecting* [3]. Similar to end-user programming [4], it recognizes upfront that the key issue is bridging the gap between available computational resources and the skill set of the users who must harness them — users who typically have low programming skills. But unlike end-user programming, it seeks to find higher-level abstractions that leverage the considerable advances in software architecture languages, methods, and tools to support component composition, analysis and execution.

While we investigated the potential of this approach across various domains — and specially the ones that already had successful end user composition platforms like Taverna and Wings in e-sciences — we found effective ecosystems consisting of domain-experts, component developers, platform developers, and end users at play. While richer compositions environments manage this synchronization quite well (although, at a great cost), several impoverished composition environments enforce end users to perform all these roles by themselves. The end user architecting approach address the problems of such end users, and the other developers in the ecosystem.

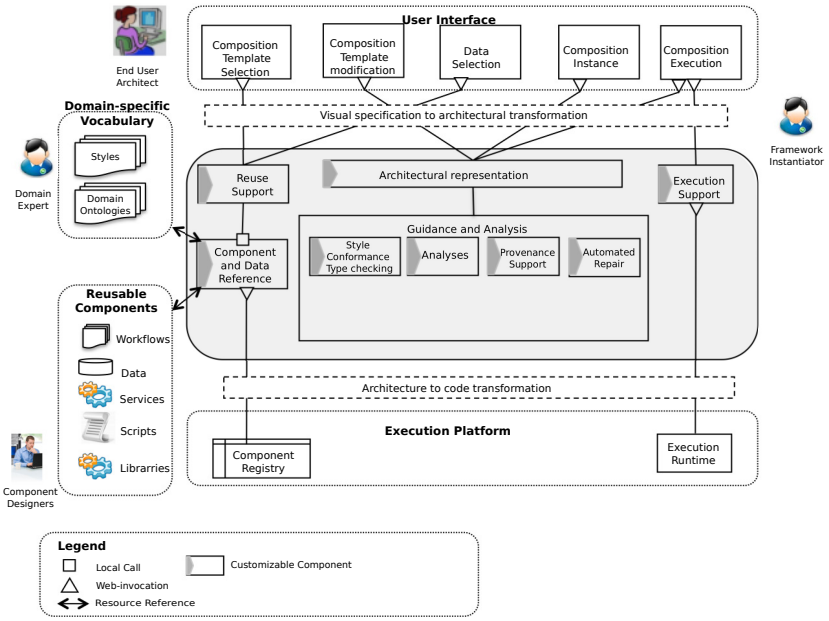

Fig. 2: Building Blocks of End User Architecting Framework

To allow an easier use of this approach, we have designed a customizable framework that can be instantiated across a large number of domains. Figure 2 shows the high-level building blocks of the framework and the larger ecosystem of developers that it helps. For preliminary investigations, we have instantiated our framework in three domains: dynamic network analysis, brain imaging, and geospatial analysis [3]. Across these domains, the end users are technically naive users (such as analysts and neuroscientists) who write compositions that are variations of dataflows and pub-sub.

Our hypothesis is that the End User Architecting framework factors out the commonalities across different classes of composition environments and computation models. Undoubtedly, some upfront cost is required to develop the composition components, the architecture styles, analyses, templates, etc. However, components, styles, templates, and analyses are reusable artifacts; thus, building libraries of them will amortize cost and reduce End User Architecting efforts for future projects.

3 Concluding remarks and future research

My current research focus is to validate the End User Architecting approach with respect to i) generality, ii) quality of compositions, and iii) reduced cost of for platform development. Apart from the three case studies mentioned in [3], I am conducting studies to determine the feasibility of applying the end user architecting approach to a larger set of domains. Examples include astronomy, bioinformatics, digital music production and other scientific computing and arts related domains where end user architecting could potentially be a good fit. Additionally, I plan to conduct usability studies to evaluate the usability and extensibility of the framework with the entire ecosystem of users in consideration.

If successful, this research will have many contributions to the field of *end user software engineering*. End user Architecting could be an effective technique for dramatically reducing the time, cost and difficulty of building a significant class of end user composition environments. This will be supported by a reusable framework that would provide interfaces, libraries, control structures and the necessary plug-in points for developing composition environments, as well as analyses that will improve end user composition experience.

Additionally, this research will also contribute to the field of *software architecture* through extensions to architecture description languages to support export and reuse of architectural specifications through generalized APIs, prior compositions, and access to repositories. The generic and reusable analytic capabilities that I am developing as a part of this research will be another significant contribution to this field.

Acknowledgements
I thank my advisor, Prof. David Garlan, for his continuing guidance and support. I also thank Bradley Schmerl, Perla Velasco Elizondo, and Ivan Ruchkin who have closely collaborated on this research. This material is based upon work funded by the Department of Defense under Contract No. FA8721-05-C-0003 with Carnegie Mellon University and Software Engineering Institute. Further support for this work came from Office of Naval Research grant ONR-N000140811223 and the Center for Computational Analysis of Social and Organizational Systems (CASOS).

References

1. Fabio Casati. How end-user development will save composition technologies from their continuing failures. In *IS-EUD*, pages 4–6, 2011.
2. Vishal Dwivedi, Perla V. Elizondo, Jose M. Fernandes, David Garlan, and Bradley Schmerl. An architectural approach to end user orchestrations. In *ECSA*, pages 370–378, 2011.
3. David Garlan, Vishal Dwivedi, Ivan Ruchkin, and Bradley R. Schmerl. Foundations and tools for end-user architecting. In *Monterey Workshop*, pages 157–182, 2012.
4. Bonnie A. Nardi. *A small matter of programming: perspectives on end user computing*. MIT Press, 1993.