Research Statement

Roberto Silveira Silva Filho
http://www.ics.uci.edu/~rsilvafi

My research so far has focused on the synergetic relationship between Groupware, Middleware and Software Engineering. As computing becomes increasingly ubiquitous, new collaborative tools and applications are created. These applications require novel software infrastructures, or Middleware, that are able to support and evolve with the constant shift of networking and application domain requirements. Novel Software Engineering fundaments and techniques are also required in support of these applications and infrastructures. A common thread in my research has been the development of collaborative tools and their supporting infrastructures, and the multi-dimensional evaluation of the trade-offs involved in their design, development and use. In particular, I focus on both end-users and developers, researching for novel software engineering principles and guidelines that can better support their activities.

My Ph.D. work at UC, Irvine, was motivated by the increasing popularity of event-driven applications and their need for application-specific features. In particular, I worked with applications in different areas including: usability monitoring [1], security [2], workspace awareness [3] and contextual collaboration [4]. These applications usually rely on custom-made publish/subscribe infrastructures, developed to meet their specific needs. This observation resulted in my work on YANCEES, a versatile publish/subscribe infrastructure [5], based on plug-ins and extensible languages, that can be extended and customized to meet the requirements of existing and novel applications. By using YANCEES, application-specific infrastructures can be built through the composition of new and existing components, thus significantly reducing the development effort of new publish/subscribe infrastructures, at the same time that better support application developers, that can rely on infrastructures that better match their application needs.

ADDING VERSATILITY TO PUBLISH/SUBSCRIBE INFRASTRUCTUES

Supporting publish/subscribe versatility. My experience in the development of YANCEES produced important insights on how to support versatility in this domain [6]. First, it showed the benefits of supporting extensibility and configurability along publish/subscribe domain main concerns. Second, it made evident the need for generalization, instead of variation, as the preferable approach to represent events, and automation, to enforce automatically allocate plug-ins, at runtime, enforcing their interdependencies. Third, it showed the benefits of separating maintenance and regular publish/subscribe APIs, on the overall usability and reuse of the infrastructure.

Analysis of YANCEES development challenges. Even though YANCEES was successful in supporting both existing and novel applications, its development faced different challenges. The first challenge was **the lack of orthogonality between the publish/subscribe variability dimensions**. These dimensions have implicit control and data dependencies that hinder their independent evolution [7]. As a result, special measures to document and enforce these dependencies are required such as the use of configuration automatic managers and the explicit declaration of dependencies in configuration files. A side effect is the increase of the complexity of the infrastructure as a whole. Note that **these dependencies are not a consequence of the variability implementation approach adopted, but are essential to the problem domain**. Second, due to implicit control and data dependencies in different variability implementation approaches (for e.g. plug-ins, filters, adapters, etc), the combined use of apparently compatible features may lead to **feature interaction**. Third, the **inadequacy of programming interfaces to document and enforce all the implicit dependencies and contracts** make it difficult for application developers to learn, customize, extend and debug the infrastructure. Information such as with what other plug-ins, timing guarantees, and event formats are plug-ins compatible with; or what's the impact of configuring and extending one feature, are not easy to grasp. Finally, the implementation of conceptually **simple features usually crosscuts different plug-ins and configuration files.** This lack of locality results in

extra cognitive effort for application developers that need to guarantee the coherent extension of different artifacts. In order to address some of these issues, complex configuration management and architecture documentation schemas were developed [8].

EMPIRICAL ANALYSIS OF VERSATILITY TRADE-OFFS IN PUBLISH/SUBSCRIBE INFRASTRUCTURES

The different problems faced in the development of YANCEES, and the existence of other, more traditional versatility approaches, motivated my Ph.D. dissertation work [9], which addresses the following key questions about the development and use of versatile publish/subscribe infrastructures: (1) How versatile are existing publish/subscribe infrastructures? (2) Do they face the same issues found in the development of YANCEES? (3) What are the common factors that hinder the development of versatile publish/subscribe infrastructures? (4) What trade-offs should developers and users consider when building, evolving and reusing these infrastructures? Answering these questions is a key step towards improving the development of versatile infrastructures in general. It also provides insights in the form of principles and guidelines that can better support infrastructure developers in building more versatile infrastructures, and application engineers in selecting the right middleware for their needs.

Comparing the versatility of existing publish/subscribe infrastructures. In order to answer these questions, I performed a multi-dimensional qualitative and quantitative analysis. This empirical study involved different publish/subscribe infrastructures, case studies and metrics. First, I selected a set of open-source publish/subscribe infrastructures representing major versatility strategies: minimal core APIs (Siena), one-size-fits-all (CORBA-NS), coordination languages (JavaSpaces) and flexible composition approaches (YANCEES). Next, I selected three heterogeneous application domains: usability monitoring, peer-to-peer file sharing, and awareness. The requirements of each application domain were abstracted in the form of ideal publish/subscribe APIs infrastructures must provide. The selected infrastructures were then adapted to match these API requirements according to three different approaches: (a) black box, where code was built around the infrastructures; and (b) grey-box, this last one only supported by YANCEES, that was extended and configured along its variation points. Finally, I measured the reusability, maintainability, flexibility, usability, and performance of each infrastructure, for each case study, using an extended metrics suite I created.

Analyzing the versatility trade-offs of existing publish/subscribe infrastructures. The analysis of versatility trade-offs revealed a set of costs and benefits of each versatility approach studied:

1. **Minimal core APIs** as Siena are efficient, have simple APIs and are easier to build than one-size-fits-all infrastructures. These infrastructures are reused by laying functionality on top of them. Through the use of generalized subscription and event representations, they can support a large set of application domains. In spite of these benefits, their core functionality is inflexible (not easily configurable or extensible), being limited by the generalized (but fixed) event, subscription and notification capabilities they provide. This approach supports black-box reuse, which results in higher middleware and adaptation costs if compared to flexible approaches as YANCEES.
2. **Coordination languages** as JavaSpaces have the similar generalization benefits of minimal core infrastructures. In the particular case of JavaSpaces, we found problems with semantic and performance mismatches, a consequence of the inflexibility of the event (tuple space) and notification models with respect to filtering capability and pull notification policy.
3. **One-size-fits-all infrastructures** as CORBA-NS support a large set of features through specialized variability and configurability. In this approach, configurability is delegated to application developers, through programmatic interfaces (for example: factories, configuration methods, and composition). This manual configurability decreases the overall system usability. This can also impact performance since the most common features end up paying the price for the extra features supported.
4. **Flexible approaches** as YANCEES may imply, in some cases, in more complex, lengthy and componentized code. This is a consequence of the generalization and separation of concerns these systems provide. However,

the resulting code is usually more modular (maintainable) and reusable. Moreover, the ability to customize the infrastructure to the application domain requirements reduces the abstraction distance between the required and provided functionality, reducing the client side development effort to build applications based on this infrastructure.

FUTURE RESEARCH

My vision is to help both infrastructure developers and users in designing, reusing, and evolving collaborative software infrastructures. From my experience in the development of YANCEES and the analysis of versatility trade-offs I conducted as part of my Ph.D. dissertation, I identified main research topics discussed as follows.

Code-based software architecture documentation and enforcement. Even though different design and implementation approaches exist to support the development of versatile software (for example, software product line engineering, component models, plug-ins, frameworks, and many others), the process of design, evolution and reuse of software developed according to these approaches is still costly and error prone. A common problem faced by users of versatile software in general is the understanding of the architectural rules and rationale used in their design. These assumptions are not always documented and not easily enforceable by existing tools. The result is architectural drift, steep learning curves, and errors driven by hidden dependencies between apparently unrelated parts of the code. In other words, the breaking of these rules are a constant threat to the architectural integrity, and over time erode the non-functional properties such as flexibility, maintainability, understandability and performance. In particular, there is a lack of usable and useful approaches to capture, represent and enforce architectural invariants in the code. By architectural invariants I mean assumptions and rules that must be followed in order to extend, adapt or maintain the code without breaking its original architectural properties.

The solution to this problem requires the gathering of otherwise hidden, scattered information, their enforcement and presentation in meaningful ways to software developers thus supporting their activity at hand. Moreover, many times, this information is tacit, not written in any documentation form, but is part of the expertise of few developers, which makes it event difficult to locate, combine and present this information. In this research, I plan on answering the following questions: What kind of architectural invariants and context do developers need? How can this information be gathered, presented and enforced? Can we derive usable and useful ways to capture, document, present and enforce this information? How can we support tacit knowledge capturing, representing and sharing? I believe that the approach must support developers in the code level, being integrated with existing IDEs and compilers.

In line with this research, I've been working on novel approaches to aspect-oriented programs understanding and evolution [10]. In this project, aspect code that usually resides in different files, is automatically weaved together with base code, and kept consistent with the program through a relational model. This strategy not only supports program comprehension, but also improves the evolution and debugging of both base and aspect codes. I plan on broadening the scope of this research to support the comprehension and evolution of other programming paradigms such as Object-Oriented programming, to address problems such as the fragile base-class problem. I also plan on integrating the tool with different information sources in support for context in software comprehension and evolution.

API usability metrics, guidelines and tool support. Application Programming Interfaces (or APIs) define reusable abstractions applied in the construction of complex software systems. They not only support the management of software complexity, but also work as boundaries between relatively independent software development teams. In spite of their importance, very little research has been done on the design and evaluation of APIs. In my dissertation work, the size of the APIs and the number of concerns API users need to master in have shown to be an important factor in the total effort of adapting, extending and configuring an infrastructure. Another important off-spring of my dissertation was a set of metrics and a tool to automatically calculate API size. This particular work has been part of a collaboration with professor Cleidson de Souza from University of Pará, Brazil. In this research, I plan on answering the following questions: What is a good API? How can we

adequately measure API usability? What's the impact of sound software engineering approaches in the resulting API usability? Can we develop better principles, guidelines and tools to better support the development of APIs?

Awareness in collaboration. One of the main themes of my research has been the adequate support for collaboration by leveraging on awareness and event-driven infrastructures and integration of information from different sources. In projects such as Continuous Coordination in distributed software engineering [3] and awareness-based security [2] I searched for evidences of the effectiveness of exposing otherwise hidden or inaccessible information in support for more effective coordination. I would like to continue my collaboration with professor David Redmiles in the study of current and novel applications of awareness such as: collaborative software engineering, software versatility and comprehension. In particular, I plan to investigate the following research questions: Can we leverage on knowledge scattered throughout different artifacts, or the expertise of different stakeholder, in support of better software engineering approaches? What kind of infrastructure support should be provided?

SELECTED REFERENCES

[1] D. Hilbert and D. Redmiles, "An Approach to Large-scale Collection of Application Usage Data over the Internet," presented at 20th International Conference on Software Engineering (ICSE '98), Kyoto, Japan, 1998.

[2] R. DePaula, X. Ding, P. Dourish, K. Nies, B. Pillet, D. Redmiles, J. Ren, J. Rode, and R. S. Silva Filho, "In the Eye of the Beholder: A Visualization-based Approach to Information System Security," *International Journal of Human-Computer Studies - Special Issue on HCI Research in Privacy and Security*, vol. 63, pp. 5-24, 2005.

[3] D. Redmiles, A. van der Hoek, B. Al-Ani, T. Hildenbrand, S. Quirk, A. Sarma, R. S. Silva Filho, C. de Souza, and E. Trainer, "Continuous Coordination: A New Paradigm to Support Globally Distributed Software Development Projects," *Wirtschaftsinformatik (Special Issue on the Industrialization of Software Development)*, vol. 49, 2007.

[4] R. S. Silva Filho, W. Geyer, B. Brownholtz, I. Guy, D. F. Redmiles, and D. R. Millen, "Architectural Trade-Offs for Collaboration Services Supporting Contextual Collaboration - RC23756," IBM T. J. Watson, Cambridge, MA RC23756, 2005.

[5] R. S. Silva Filho, C. R. B. de Souza, and D. F. Redmiles, "The Design of a Configurable, Extensible and Dynamic Notification Service," presented at International Workshop on Distributed Event Systems (DEBS'03), San Diego, CA, 2003.

[6] R. S. Silva Filho and D. Redmiles, "Striving for Versatility in Publish/Subscribe Infrastructures," presented at 5th International Workshop on Software Engineering and Middleware (SEM'2005), co-located with ESEC/FSE'05 Conference, Lisbon, Portugal, 2005.

[7] R. S. Silva Filho and D. F. Redmiles, "Towards the use of Dependencies to Manage Variability in Software Product Lines," presented at Workshop on Managing Variability for Software Product Lines. (SPLC'2006), Baltimore, MD, 2006.

[8] R. S. Silva Filho and D. F. Redmiles, "Managing Feature Interaction by Documenting and Enforcing Dependencies in Software Product Lines," presented at 9th International Conference on Feature Interaction, Grenoble, France, 2007.

[9] R. S. Silva Filho and D. F. Redmiles, "An Analysis of Publish/Subscribe Middleware Versatility," Institute for Software Research, Irvine, CA UCI-ISR-09-3, August 2009.

[10] W. Ruengmee, R. S. Silva Filho, S. K. Bajracharya, D. F. Redmiles, and C. V. Lopes, "XE (eXtreme Editor) - Bridging the Aspect-Oriented Programming Usability Gap," presented at Automated Software Engineering, 2008. ASE 2008. 23rd IEEE/ACM International Conference on, 2008.