Relating Patterns and Reference Architectures

EDUARDO GUERRA, National Institute for Space Research (INPE), Brazil
ELISA YUMI NAKAGAWA, University of São Paulo (ICMC/USP), Brazil

Both patterns and reference architectures aim to describe solutions to be reused for the software systems development. Despite that, they have a lot of differences and have been investigated separately. The objective of this paper is to discuss the relationship between them, how they can be complementary, while respecting their respective peculiarities. We also discuss how patterns can support the creation of reference architectures and how reference architectures can be a source for pattern mining.

Categories and Subject Descriptors: D.1.5 [**Programming Techniques**] Object-oriented Programming; D.2.11 [**Software Architectures**] Patterns

General Terms: Reference Architecture

Additional Key Words and Phrases: Reference architecture, Software Architecture, Patterns

ACM Reference Format:

Guerra, E. and Nakagawa, E. , 2015. Relating Patterns and Reference Architectures. HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 22 (October 2015), 9 pages.

1. INTRODUCTION

Patterns are recurrent solutions used for problems on different software systems that share a specific context. In the field of software design and architecture, patterns present technical solutions for how to structure software components to achieve desired requirements. On the other hand, a reference architecture is a special type of software architecture that can be used as a basis for the construction, standardization, and evolution of software architectures of systems in a given domain. It defines elements, their roles and relationships needed to solve common problems in targeted domains. Based on their description, it is possible to perceive that these two ways of documenting reusable solutions have a lot in common with each other. However, despite few isolated works that focus on specific domains [Stricker et al. 2010] [Guerra et al. 2013a] [Fernandez et al. 2015], this relationship is not being explored. The goal of this paper is to explore the existing relationships between patterns and reference architectures, and present a discussion to work with both in a complementary way and, as a consequence, to more effectively support the development and design of software systems.

The target audience for this paper is both pattern and reference architecture specialists. On the one hand, it is expected for a pattern expert to understand better how pattern languages can be a basis for reference architectures and how they can be a source of information for pattern mining. On the other hand, reference architecture specialists can understand how they can get benefit from patterns in the design of reference architectures.

This work is supported by São Paulo Research Foundation (FAPESP, Grant: 2014/02244-7 and 2015/16487-1), CNPq (Grant: 445562/2014-5) and Brazilian funding agency Capes (Grant: 2012/034).
Author's emails: E. Guerra, eduardo.guerra@inpe.br; E. Nakagawa, elisa@icmc.usp.br

The paper is organized as follows: Section 2 presents an overview of reference architecture concepts and Section 3 an overview of patterns and related concepts; Section 4 highlights differences between patterns and reference architectures; Section 5 focuses on guidelines for using a pattern language as the basis for reference architectures; Section 6 presents some research perspectives about using reference architectures for pattern mining; Section 7 describes an existing process for reference architecture creation and presents a proposal to introduce patterns usage on it; and, finally, Section 8 concludes the paper.

2. AN OVERVIEW OF REFERENCE ARCHITECTURES

The goal of this section is to describe what a reference architecture is, what its characteristics are and provide some examples of existing ones. This information is relevant to compare them with patterns.

A reference architecture refers to an architecture that encompasses the knowledge about how to design concrete architectures of systems of a given application domain; therefore, it must address the business rules, architectural styles (sometimes also defined as architectural patterns that can also address quality attributes in the reference architecture), best practices of software development (for instance, architectural decisions, domain constraints, legislation, and standards), and the software elements that support development of systems for that domain. All of this must be supported by unified, unambiguous, and widely understood domain terminology [Nakagawaa et al. 2015].

Reference architectures usually contain important knowledge about how to organize architectures of software systems in a given domain, presenting several possibilities of use. Diverse purposes have guided their adoption and use, such as the creation of concrete architectures to support the building of software systems, standardization of systems of a given domain, evolution of existing software systems, construction of new reference architectures, and support for the building of software product line.

A classification based on their goal divide reference architectures in two types [Angelov et al. 2012]: for standardization and for facilitation. On the one hand, the standardization of concrete architectures aims to provide system and component interoperability, and usually it is defined based on a set of existing architectures. On the other hand, the facilitation of the design of concrete architectures aims to provide guidelines and inspiration for the design of systems. This second one, despite can be defined for a domain with well established architectures, can also be defined for new domains that do not have still reference architectures.

Considering their relevance, different domains have already understood the need for encapsulating knowledge in reference architectures, with the aim of disseminating and reusing this knowledge and standardizing the systems as well. Good examples are AUTOSAR (AUTomotive Open System ARchitecture) [Partnership 2015], for the automotive domain, and Continua [Alliance 2013a] and UniversAAL [PROJECT 2013], for AAL (Ambient Assisted Living). In particular, these architectures have been developed by consortiums that involve major industrial players (such as manufacturers and suppliers) and researchers.

Besides that, platform-oriented architectures, i.e., architectures not related to a specific application domain, but to a specific architectural style or technology, have also been proposed and widely used as reference architectures. Some good examples are OASIS [OASIS 2008] and S3 [Arsanjani et al. 2007], for software systems based on SOA (Service-Oriented Architecture) [Papazoglou et al. 2008], and OSGi (Open Services Gateway initiative framework) [Alliance 2013b], a set of specifications and a layered architecture that define dynamic component systems for Java. However, it is observed that these architectures were created without using a systematic approach, what can be consuming extra effort, time, and even requiring rework.

In order to systematize the establishment of reference architectures, some initiatives can be found, such as [Angelov et al. 2012] [Bayer et al. 2003] [Cloutier et al. 2010] [Dobrica and NiemelÃď 2008] [Galster et al. 2011] [Muller 2008]. They are in general high-level guidelines, principles, recommendations or domain-specific processes. In this scenario, we proposed ProSA-RA [Nakagawaa et al. 2015], a process for the building of reference architectures, focusing on how to design, represent, and evaluate such architectures. It has been widely used to create several architectures for different domains and scenarios (both industry and academia); however, we had not still explored the use and benefits that could be provided by patterns.

3. AN OVERVIEW OF PATTERNS AND RELATED CONCEPTS

Patterns can be defined as recurring solutions that can be reused in different contexts. Despite patterns were used in other field before [Alexander et al. 1977], they became popular in the software development community with the book "Design Patterns: Elements of Reusable Object-Oriented Software" [Gamma et al. 1995]. Today, patterns are widely used by software developers and architects as a reference in software design.

The documentation of patterns usually uses a standard format, whose core elements are usually context, problem and solution. Other important pattern elements are forces, solution description, consequences, related patterns and known uses. Other sections complementary sections can also be included, such as implementation details and examples, as well sections related to the pattern domain.

Patterns can be related to other patterns in different ways [Buschmann et. al 2007]. A Pattern Collection can be used to organize and classify patterns related to a common domain. A Pattern Compound, also known as Compound Pattern, can document a recurrent combination of patterns, which can be considered itself a pattern. Moreover, a pattern language, which focuses on how patterns relate in a given domain, is a strongest relation that patterns can have, documenting a broader knowledge in that context. Despite that, the term "pattern language" is commonly used by the pattern community, there is no formal boundary that divides pattern collections from pattern languages. Given this context, this paper refers to pattern language as a set of related patterns.

4. DIFFERENCES BETWEEN PATTERNS AND REFERENCE ARCHITECTURES

This section aims to compare patterns and reference architectures in several aspects, such as information novelty, specificity, relation to application domain, documentation, and update. As a reference architecture is a broader solution, in some cases, it is more fair to compare such architecture to a set of patterns instead of a single one.

An important difference between a pattern and a reference architecture is that a pattern must be a recurrent solution already used in existing implementations. On the other hand, a reference architecture, specially the facilitation one [Angelov et al. 2012], can propose innovative solutions in its structure, whose usage was not yet proven in existing software. As a consequence, a reference architecture is more suitable for proposing a structure for new domains, which are not still well established. Moreover, for a domain that already has many applications implemented, it is interesting to examine such reference architecture, but it is necessary to identify the recurrent solutions used by these applications.

Patterns and reference architectures document solutions differ at levels of specificity. While a pattern focuses on a single problem and in a single recurrent solution, a reference architecture usually considers the target domain as a whole. Hence, a pattern language or other kinds of pattern grouping are closer to the solution scope of a reference architecture. However, even a pattern language does not have explicit requirements to fulfill and is not necessarily complete, it documents recurrent solutions that are already well established for a specific domain.

A pattern can be more general or more specific to a given domain, depending on the problem in which it is related to and the scope where the pattern mining was performed. It can be fine-grained or larger. On the other hand, a reference architecture is commonly focused on a single domain, containing patterns that can be appropriate for other architectures, too.

A pattern is documented based on a template composed by sections, which are at least its context, its problem, and its solution. Other common sections present the forces, consequences, alternatives for implementations, and examples of use. In a pattern language, the patterns usually have a standard structure. A reference architecture has a more comprehensive documentation, usually with many pages, including models, tables, texts, and in some rare cases a reference implementation.

When we consider evolution, a fast way to improve a pattern language is by adding new pattern to it. Being a recurrent solution, this new pattern can complement the existing knowledge in different ways, either by offering an alternative solution or by adding a solution to a different problem. However, this addition should be done carefully, avoiding redundancy and confusion. However, despite it is common to update a pattern language by adding new patterns, rarely pattern authors go back and update individual patterns already documented. When evolving a reference architecture, a new version is released. This version can present minor or major additions, but it should be released as a whole. Table I summarizes the differences between both of them. The column related to patterns presents characteristics for a single pattern and a pattern language, when applicable.

Table I. Summary of Differences between Patterns and Reference Architectures

Characteristic	**Single Pattern**	**Pattern Language**	**Reference Architecture**
Information novelty	Recurrent solution	Recurrent solution	Innovative solution (facilitation) or recurrent solution
Specificity	Single problem	Related problems in a shared domain	Multiple related problems
Relation to application domain	Independent of or dependent on the application domain	ndependent of or dependent on the application domain	Dependent on the application domain in most cases
Documentation	Pattern description, usually using a standard pattern template	A diagram relating the patterns	Many pages, including models, tables, texts, and even implementation
Update	New implementation strategies and variations; Rarely updated.	Additions of new patterns	Minor or major additions considering the whole document

5. FROM A PATTERN LANGUAGE TO A REFERENCE ARCHITECTURE

The goal of this section is to develop guidelines for creating a reference architecture based on a pattern language. It is worth highlighting that when reference architectures are created, several information sources must be considered, including reference models, domain models, conceptual models, domain ontologies, and so on. In particular, our proposal explicitly explores the use of pattern languages and patterns for that purpose. Reference architectures can be also built out from analysis/design patterns; moreover, other patterns, such as for security, safety, or reliability, can be combined into such architectures.

Our guidelines can also be followed to apply other types of pattern groupings such as patterns collections to contribute to the reference architecture building. It is important to highlight that a reference architecture does not necessarily need to be completely based on patterns. For problems in which the solution is not consolidated in existing implementations, some ad-hoc solutions can also be used.

As stated before, a pattern language is a set of related patterns that capture recurrent solutions for common problems in a given domain. Despite patterns can document relations and dependencies with other patterns in the language [Noble 1998], a pattern language does not present a single structure. To create a reference architecture based on a pattern language, it is important to show how the patterns fit together to support it.

A pattern usually present components (often described in a "Participants" section), which interact to solve a target problem. In a pattern language, patterns handle problems in the same domain. Hence, it is common to have components that play the same role in different patterns of the language. Based on that, the structure can be unified by using the common components as points to bind the pattern solutions together.

When there is some patterns or group of patterns that end up without a connection to other patterns, there can be some missing pattern on the pattern language. That can be a problem, because the participants need to be bound to a single structure in the reference architecture. One option is to perform some pattern mining on the existing systems to specifically search for solutions that binds to the reference structure. If the solution used by the systems is not uniform, in other words, there is not a consolidated pattern, a new or a particular solution can be proposed and used in the reference architecture. Pattern solutions can also be complemented by new proposals or solutions inspired in more general patterns.

When a pattern language presents alternative patterns for the same problem, one of them should be chosen to be used in the reference architecture. Despite having a discussion on how to add this kind of variability to reference architectures [Nakagawa 2012], it is not common to find alternative solutions in existing reference architectures. An alternative to add both solutions is to define variabilities in the reference architecture, representing different implementations. However, supporting both solutions in the same structure might not represent faithfully the domain's needs, since only one of them should be used.

It is important to highlight that these ways for identifying missing patterns or alternative solutions to define reference architectures from a pattern language need deeper investigations and validation. The presented approaches are proposals of how they can be handled.

An example of use of this proposed processes happened in the reference architecture for metadata based frameworks [Guerra et al. 2013a]. This reference architecture was created based on a pattern language [Guerra et al. 2013b] that documents eight patterns focused on recurrent solutions for metadata processing and reading used in this kind of framework. In this case, the patterns in the pattern language were enough to bind all the components referenced on the patterns.

The same happened in the Adaptive Object Models (AOM) domain. Initially, some patterns documented practices on how to create adaptive objects using metadata [Foote and Yoder 1998]. After that, it evolved to what was called an architectural style [Yoder and Johnson 2002], which is way closer to a reference architecture. Since it is easier to add extensions to a set of patterns, follow on work focused on other aspects of the AOM architecture, such as graphical interface rendering [Welicki et al. 2007] and the creation of objects [Welicki et al. 2010]. Despite these extensions, a new reference architecture was not defined nor the architectural style was redefined, since these new patterns were meant to be used with the initial AOM patterns, they are perfectly compatible with the documented AOM architectural style. Other similar examples are the reference architecture for service-based systems [Stricker et al. 2010] and a security reference architecture for cloud systems [Fernandez et al. 2015].

6. USING REFERENCE ARCHITECTURES FOR PATTERN MINING

The goal of this section is to present a future perspective of how reference architectures can be used for pattern mining. It is important to highlight that this proposal was not evaluated yet and should be considered just as a proposal of this approach.

Pattern mining consists of searching for recurrent solutions in a target domain. For architectural design and, more generally, software patterns, this search can be conducted by using code inspection. Software documentation, such as diagrams and descriptions, can also be used in this activity. This search can be focused on the problem, looking at how a software solves a particular issue. The pattern mining can be also based on the structure of several software applications related to the same domain, searching for common practices. In both activities, the reference architecture can be used as a source for the pattern mining. It can be compared to actual implementations to identify if the practices proposed by the reference architecture are actually used on real software. A reference architecture that was already used on several projects can also be a source of information.

7. REFERENCE ARCHITECTURE CREATION

Similar to processes for constructing software architectures, reference architectures should be built using a set of systematized steps, as already discussed by the software architecture community. This section is organized as follows: 7.1 aims to present one process that has supported the building of reference architectures [Nakagawaa et al. 2015], and 7.2 presents a proposal of how patterns might be introduced in such process.

7.1 Existing Reference Architecture Creation Process

From our experience in working with reference architectures, as well as in systematizing the steps to build them [Nakagawa et al. 2012] [Nakagawa 2012] [Nakagawaa et al. 2015] [Guerra et al. 2013a], we can summarize the whole building process as presented in Figure 1. To adequately apply this process, first, it is specially important to establish the scope of the reference architecture, i.e., the target application domain. This scope can be defined using the set of systems that are intended to be produced based on this architecture. In short, four steps compose this process:

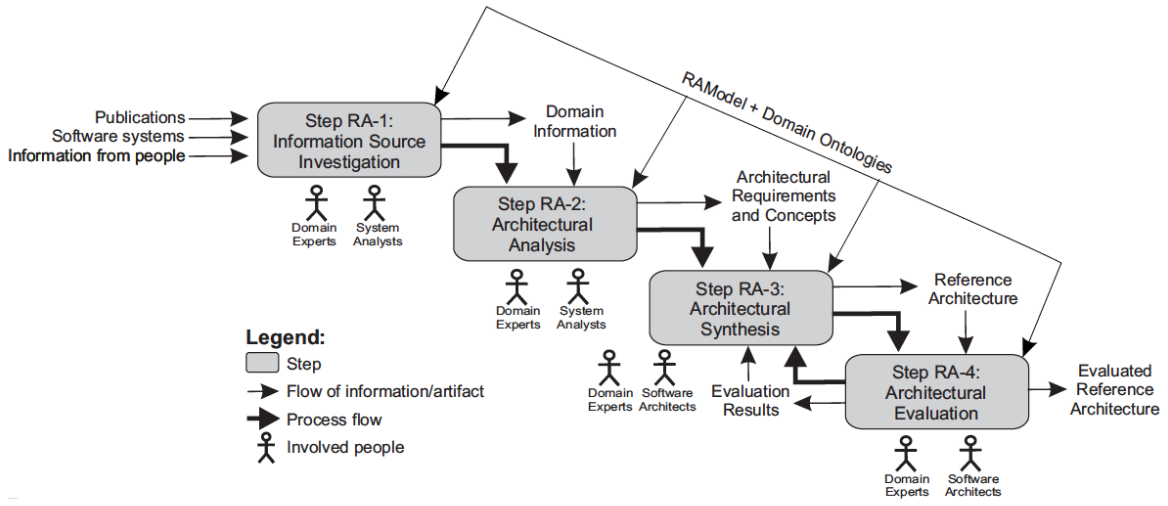
Publications
Software systems
Information from people
Step RA-1: Information Source Investigation
Domain Information
Domain Experts
System Analysts
Step RA-2: Architectural Analysis
Architectural Requirements and Concepts
RAModel + Domain Ontologies
Step RA-3: Architectural Synthesis
Reference Architecture
Software Architects
Evaluation Results
Step RA-4: Architectural Evaluation
Evaluated Reference Architecture
Legend:
Step
Flow of information/artifact
Process flow
Involved people

Fig. 1. Outline Structure of ProSA-RA

—Step RA-1: Information Source Investigation: In this step, the main information sources are selected. These sources must provide information about processes and activities that can be supported by software systems of the target domain. Since reference architectures should be the basis for several software systems in a given domain, these sources must involve a more comprehensive knowledge about the domain, than compared to information sources used when developing the architecture of a specific system. From this perspective, ProSA-RA highlights the most relevant sources: people, documents, systems of the domain, related reference models and reference architectures, and domain ontologies, if available;

—Step RA-2: Architectural Analysis: Based on the selected sources, three set of elements are identified. First, the set of requirements of software systems of that domain is identified and, based on these requirements, the set of requirements for the reference architecture is then identified. These requirements include functional and non-functional requirements. After that, the set of concepts that must be considered in this reference architecture is established. For this, three main tasks are performed: (i) identification of the system requirements; (ii) identification of the reference architecture requirements; and (iii) identification of the domain concepts;

—Step RA-3: Architectural Synthesis: In this step, the architectural description of the reference architecture is built following a model for reference architectures, such as RAModel [Nakagawa et al. 2012]. For instance, architectural styles, as well as a combination of these and other styles, identified in Step RA-1, must be considered. These styles are the basis on which concepts previously identified are organized. For instance, if a three-tier architecture is used as an architectural style of the reference architecture, the concepts related to the business rules should be organized into the application tier, since this tier usually contains these rules. Considering the effectiveness of using architectural views to represent concrete software architectures, they can be also adopted to describe reference architectures; and

—Step RA-4: Architectural Evaluation: In the context of this work, reference architecture evaluation refers to the task of reviewing the architectural description together with diverse stakeholders intending to detect defects in this description. For this, ProSA-RA uses a checklist-based inspection approach, called FERA (Framework for Evaluation of Reference Architectures). In short, FERA is composed of 93 multiple choice questions whose answers vary from “fully satisfactory” to “totally unsatisfactory”, and fields to add comments. It can be evaluated by a range of stakeholders (namely architects, domain experts, analysts, software project manager, designers/developers/implementers, integrators, testers, and quality assurance stakeholders).

7.2 Pattern-based Method for Reference Architecture Creation

Aiming at exploring the benefits of using recurrent, well-tested solution during the construction of reference architectures, patterns become quite interesting to be included in ProSA-RA. In particular, Step RA-3 can benefit by these patterns, what includes mainly design patterns and architectural patterns. However, depending on the reference architecture domain, other kinds of patterns can also be considered, such as security patterns, safety patterns, and testing patterns.

During this step, there are several sources for patterns that can be used, such as patterns repositories, pattern collections or pattern languages related to the reference architecture domain, as well as patterns related to a problem domain that is facing the reference architecture (security, reliability, distribution, etc...). A particular good source can be also specialists in patterns or domain specialists. The patterns can be used in the following activities:

—Identification of a set of candidate patterns. For this, requirements for the reference architecture identified in the last step must be considered;
—Selection of patterns that fit the needs of the reference architecture. Benefits and drawbacks of adopting each pattern must be investigated separately;
—Joint analysis of all patterns that are intended to be adopted;
—Decision of the Points of Application (PA), i.e., part/component of the reference architecture where each pattern will be applied and also when two or more patterns are concurrent in the same PA. This last issue characterizes the variation points of the reference architecture regarding pattern application; and
—Design of the reference architecture by creating architectural views and models that include the selected patterns.

Besides using patterns for these architectures, the use of pattern languages are also aligned with goals of such architectures.

When the search does not find a pattern for a given problem related to a reference architecture, an alternative is to conduct a pattern mining activity to find a pattern on existing systems. This pattern can be documented and stand on its own, and can be further be incorporated in the reference architecture.

8. CONCLUSION

Patterns, pattern languages, and reference architectures must be investigated in a complementary way to provide a more complete solution to the development of software systems. Initiatives in that direction can be already found and there are still good open research perspectives. The use of patterns and pattern languages to create reference architecture could bring important contributions, such as reduction in time and efforts to design reference architectures and improvement in the reference architecture quality, as parts of such architecture adopt proven solutions. It is important to highlight that both areas (patterns and reference architecture) have already achieved a consolidated status, with recognized advantages and solutions; therefore, it seems natural to investigate them together, contributing to the advance of the state of the art of these areas.

9. ACKNOWLEDGMENTS

This study is supported by SÃčo Paulo Research Foundation (FAPESP, Grant: 2014/02244-7 and 2015/16487-1), CNPq (Grant: 445562/2014-5) and Brazilian funding agency Capes (Grant: 2012/034). We would also like to thank Prof. Dr. Eduardo Fernandez from Florida Atlantic University (FAU) who served as a shepherd, providing us valuable comments that led us to a considerable improvement of the paper. We also recognize and thank the contributions given from the Writers Workshop participants.

REFERENCES

Christopher Alexander, Sara Ishikawa, and Murray Silverstein. 1977. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, New York. http://www.amazon.fr/exec/obidos/ASIN/0195019199/citeulike04-21

Continua Health Alliance. 2013a. Continua Health Alliance. http://www.continuaalliance.org/. (2013).

OSGi Alliance. 2013b. OSGi Alliance Specifications. http://www.osgi.org/Specifications/. (2013).

Samuil Angelov, Paul Grefen, and Danny Greefhorst. 2012. A Framework for Analysis and Design of Software Reference Architectures. *Inf. Softw. Technol.* 54, 4 (April 2012), 417–431. DOI:http://dx.doi.org/10.1016/j.infsof.2011.11.009

Ali Arsanjani, Liang-Jie Zhang, Michael Ellis, Abdul Allam, and Kishore Channabasavaiah. 2007. S3: A Service-Oriented Reference Architecture. *IT Professional* 9, 3 (2007), 10–17. DOI:http://dx.doi.org/10.1109/MITP.2007.53

Joachim Bayer, Dharmalingam Ganesan, Jean-Francois Girard, Jens Knodel, Ronny Kolb, and Klaus Schmid. 2003. *Definition of Reference Architectures based on Existing Systems*. IESE-Report 085.03/E. Fraunhofer IESE, Germany.

Robert Cloutier, Gerrit Muller, Dinesh Verma, Roshanak Nilchiani, Eirik Hole, and Mary Bone. 2010. The Concept of Reference Architectures. *Syst. Eng.* 13, 1 (Feb. 2010), 14–27. DOI:http://dx.doi.org/10.1002/sys.v13:1

Liliana Dobrica and Eila NiemelÃď. 2008. An Approach to Reference Architecture Design for Different Domains of Embedded Systems.. In *Software Engineering Research and Practice* (2009-02-09), Hamid R. Arabnia and Hassan Reza (Eds.). CSREA Press, 287–293. http://dblp.uni-trier.de/db/conf/serp/serp2008.html#DobricaN08

Eduardo B. Fernandez, Raul Monge, and Keiko Hashizume. 2015. Building a security reference architecture for cloud systems. *Requirements Engineering* (2015), 1–25. DOI:http://dx.doi.org/10.1007/s00766-014-0218-7

B. Foote and J. Yoder. 1998. Metadata and Active Object-Models. (1998). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.1042

Matthias Galster, Paris Avgeriou, Danny Weyns, and Tomi Männistö. 2011. Variability in Software Architecture: Current Practice and Challenges. *SIGSOFT Softw. Eng. Notes* 36, 5 (Sept. 2011), 30–32. DOI:http://dx.doi.org/10.1145/2020976.2020978

Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1995. *Design Patterns: Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Eduardo Guerra, Felipe Alves, Uirá Kulesza, and Clovis Fernandes. 2013a. A Reference Architecture for Organizing the Internal Structure of Metadata-based Frameworks. *J. Syst. Softw.* 86, 5 (May 2013), 1239–1256. DOI:http://dx.doi.org/10.1016/j.jss.2012.12.024

Eduardo Guerra, Jerffeson Souza, and Clovis Fernandes. 2013b. *Transactions on Pattern Languages of Programming III*. Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Pattern Language for the Internal Structure of Metadata-Based Frameworks, 55–110. DOI:http://dx.doi.org/10.1007/978-3-642-38676-3_3

Gerrit Muller. 2008. A Reference Architecture Primer. http://www.gaudisite.nl/. (2008).

Elisa Y. Nakagawa. 2012. Reference Architectures and Variability: Current Status and Future Perspectives. In *Proceedings of the WICSA/ECSA Workshop âĂŹ12*. 159–162.

Elisa Y. Nakagawa, Flavio Oquendo, and Martin Becker. 2012. RAModel: A Reference Model of Reference Architectures. In *Proceedings of the WICSA/ECSA âĂŹ12*. 297âĂŞ301.

Elisa Y. Nakagawaa, Flavio Oquendo, and Jose C. Maldonado. 2015. *Software Architecture: Principles, Techniques, and Tools*. John Wiley Sons, Chapter Reference Architectures, 101–122.

J. Noble. 1998. Classifying relationships between object-oriented design patterns. In *Software Engineering Conference, 1998. Proceedings. 1998 Australian.* 98–107. DOI:http://dx.doi.org/10.1109/ASWEC.1998.730917

OASIS. 2008. Reference Architecture for Service Oriented Architecture Version 1.0. (2008).

Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. 2008. Service-Oriented Computing: A Research Roadmap. *International Journal of Cooperative Information Systems (IJCIS)* 17, 2 (June 2008), 233–255. DOI:http://dx.doi.org/10.1142/S0218843008001816

AUTOSAR Development Partnership. 2015. AUTOSAR (AUTomotive Open System ARchitecture). http://www.autosar.org/. (2015).

UNIVERSAAL PROJECT. 2013. The UniversAAL Reference Architecture. http://www.universaal.org/. (2013).

Vanessa Stricker, Kim Lauenroth, Piero Corte, Frederic Gittler, Stefano De Panfilis, and Klaus Pohl. 2010. Creating a Reference Architecture for Service-Based Systems - A Pattern-Based Approach. In *Towards the Future Internet - Emerging Trends from European Research*. 149–160. DOI:http://dx.doi.org/10.3233/978-1-60750-539-6-149

León Welicki, Joseph W. Yoder, and Rebecca Wirfs-Brock. 2007. Rendering Patterns for Adaptive Object-models. In *Proceedings of the 14th Conference on Pattern Languages of Programs (PLOP '07)*. ACM, New York, NY, USA, Article 12, 12 pages. DOI:http://dx.doi.org/10.1145/1772070.1772085

León Welicki, Joseph W. Yoder, and Rebecca Wirfs-Brock. 2010. Adaptive Object-model Builder. In *Proceedings of the 16th Conference on Pattern Languages of Programs (PLoP '09)*. ACM, New York, NY, USA, Article 4, 8 pages. `DOI:http://dx.doi.org/10.1145/1943226.1943231`

Joseph W. Yoder and Ralph Johnson. 2002. The adaptive object-model architectural style. Kluwer, B.V, 3–27.

Received May 2015; revised September 2015; accepted February 2015