Enhancing Safety and Security of Distributed Systems through Formal Patterns*

Jonas Eckhardt[1,2,3], Tobias Mühlbauer[1,2,3]
Supervisors: José Meseguer[4], Martin Wirsing[1]

[1] Ludwig Maximilian University of Munich, [2] Technical University of Munich,
[3] University of Augsburg, [4] University of Illinois at Urbana-Champaign

Abstract. Distributed systems are often safety- and security-critical systems and have strong qualitative and quantitative formal requirements, equally important time-critical performance-based quality of service properties, and need to dynamically adapt to changes in a potentially hostile and often probabilistic environment. These aspects make distributed systems complex and hard to design, build, test, and verify. To tackle this challenge, we propose a formal pattern-based approach and framework for the design of correct-, secure-, and safe-by-construction distributed systems.

Key words: formal patterns, meta-object pattern, statistical model checking, rewriting logic, distributed systems, cloud computing

1 Introduction

On June 20, 2011, the Cloud-based file storage service Dropbox reported that "Yesterday we made a code update at 1:54pm Pacific time that introduced a bug affecting our authentication mechanism. We discovered this at 5:41pm and a fix was live at 5:46pm." [4]. During these nearly four hours, the broken authentication mechanism granted access to possibly private data stored on some accounts using any chosen password. Issues like this are not the exception which is also reflected by the list of the top 10 obstacles for the adoption and growth of Cloud Computing [5]; with data confidentiality and auditability, availability of service, and bugs in large distributed systems being obstacles on this list. In fact, distributed systems (i) are safety- and security-critical systems which have strong qualitative and quantitative formal requirements, (ii) have equally important time-critical performance-based quality of service properties (e.g., availability), and (iii) need to dynamically adapt to changes in a potentially hostile (e.g., distributed denial of service attacks) and often probabilistic environment they operate in. These aspects make distributed systems complex and hard to design, build, test, and verify.

Modular approaches tackle the aforementioned complexity in the early stages of system design and analysis. These approaches include design patterns, which are general, reusable solutions to commonly occurring software problems and

* This work has been partially sponsored by the Software Engineering Elite Graduate Program and the EU-funded project FP7-256980 NESSoS.

have been successfully used in different domains including object-oriented software design [10], service-oriented computing [12,9] and security [16]; they clearly define the programming context, the problem, and the advantages and disadvantages of the design solution (see e.g., [10,16]).

In addition to "normal" design patterns, formal patterns are reusable solutions that are formally specified with precise semantic requirements and come with strong formal guarantees. Distributed systems can be specified as compositions of instances of such formal patterns.

Research Goals and Contributions. The main goal of the proposed research is to contribute a formal pattern-based approach and framework for the design of correct-, secure-, and safe-by-construction distributed systems, aided by a rich tool environment. The approach is based on the ideas of (i) developing executable formal models of distributed system designs, (ii) making these designs modular based on highly reusable formal patterns, and (iii) formally analyzing such models to verify qualitative (e.g., invariants) and quantitative (e.g., expected throughput) properties. This approach distinguishes itself by using executable formal pattern-based system specifications and statistical model checking, which allows the verification of larger system instances than with conventional model checking techniques (state explosion).

Rewriting logic and Maude. In order to specify executable formal patterns, an appropriate semantic framework is needed. We chose rewriting logic [13], a simple, yet powerful, computational logic and a general formalism that is a natural model of computation and an expressive semantic framework for concurrency, parallelism, communication, interaction, and object-orientation. It is capable of logical and distributed object reflection and, through its probabilistic [2] and real-time extensions [15], of modeling real-time, stochastic, and hybrid systems.

Maude [7] is a high-performance implementation of rewriting logic capable of executing rewriting logic-based specifications. The key concept of Maude specifications is that of a module. *Object-oriented modules* define objects, their state, and messages; where objects communicate via asynchronous or synchronous message passing. Distributed systems are modelled by object-oriented modules, where the state of such a system is a multiset or "soup" of objects and messages, called a *configuration*. A *parameterized module* $M[X :: P]$ has a formal parameter X satisfying a parameter theory P; M can be instantiated by another module Q via a theory interpretation $V : P \longrightarrow Q$, called a *view*, with the usual pushout semantics (see [7]). We denote the resulting module by $M[V]$ or shorter by $M[Q]$ if V is clear from the context.

The Maude system has an extensive tool environment which, e.g., includes a LTL model checker (see [7]) and the statistical model checker PVeStA [3]. The Maude system and its tool environment are the foundation of our work.

Outline. This paper proceeds as follows: Sect. 2 introduces the concept of formal patterns and gives the example of the general meta-object pattern. In Sect. 3 we discuss our proposed approach for the design of correct-, secure-, and safe-by-construction distributed systems in more detail. In Sects. 4 and 5, we respectively discuss a research plan for future work and summarize our results.

2 Formal Patterns

Formal patterns [8] enhance pattern descriptions with formal executable specifications that can support the mathematical analysis of qualitative and quantitative properties. Just as "normal patterns", a formal pattern Pat is structured in context, problem, solution, advantages and shortcomings (cf. e.g. [16,9]). Instead of using UML or Java we describe these patterns formally as a paramterized module $M[S]$ with a parameter theory S in Maude. The context of the pattern typically includes a description of the assumptions of the parameter theory S. Many of the advantages and shortcomings of the formal pattern can be gained from formal analyses.

Two formal patterns Pat and Pat' can be composed by the pattern composition $Pat + Pat'$. The problem statement and context of $Pat + Pat'$ can be systematically derived from those of Pat and Pat'. As we will see, such a composition of patterns might combine advantages while cancelling out disadvantages.

Example: The Meta-Object Pattern. Many distributed systems need to function in potentially hostile environments such as the Internet. Additionally, safety, real-time and quality of service requirements need to be satisfied. Modularization is an instrument that helps the designer or architect to cope with the high complexity of such a system. The Meta-Object (MO) pattern provides an approach based on modularization. It is defined as follows:

Context. A concurrent and distributed object-based system.

Problem. How can the communication behavior of one or several objects be dynamically *mediated/adapted/controlled* for some specific purposes?

Solution. A meta-object is an object which dynamically *mediates/adapts/controls* the communication behavior of one or several objects under it. In rewriting logic, a meta-object can be specified as an object with an inner configuration that contains the object or objects that are controlled by the meta-object. Thus, the parameterized module $MO[X]$ introduces the meta-object constructor; the parameter X specifies the controlled system.

Advantages and Shortcomings. MO defines a general control and wrapper architecture; but may add communication indirection and the requirement for language specific object visibility.

MO is a widely used pattern: The meta-object is sometimes called an *onion-skin* meta-object [1] if the inner configuration contains a single object, which itself could be wrapped inside another meta-object, and so on, like the skin layers in an onion. More generally, the inner configuration may not only contain several objects $o_1 \ldots, o_m$ inside: it may also be the case that some of these o_i are themselves meta-objects that contain other objects, which may again be meta-objects, and so on. That is, the more general reflective meta-object architectures are so-called "Russian Dolls" architectures [14].

Figure 1 illustrates the idea of a hierarchical composition of meta-objects and components according to the Russian Dolls model with boundary-crossing messages M, M', and M''. Messages addressed to the internal components $C_1 \ldots C_N$ first need to cross the boundaries of the two outer meta-objects MO_1 and MO_2'.

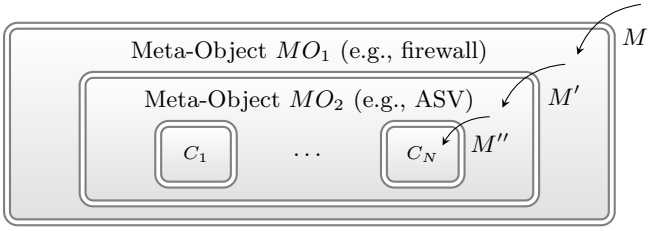

Fig. 1. Example of a hierarchical composition of meta-objects and components according to the Russian Dolls model with boundary-crossing messages.

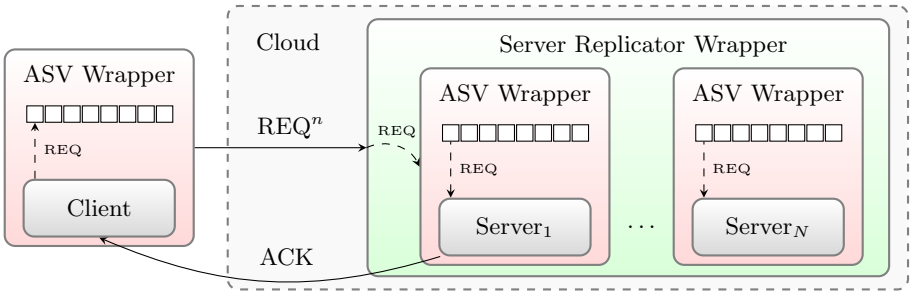

Fig. 2. Application of the ASV$^+$SR meta-object composition on a Cloud-based client-server request-response service.

The outermost meta-object MO_1 may thereby be a firewall that forwards selected messages to its inner configuration according to specific filter rules, and the inner meta-object MO_2 may be a **D**istributed **D**enial **o**f **S**ervice (DDoS) defense mechanism like the ASV protocol [11].

3 Approach: Enhancing Safety and Security through Formal Patterns

In [8], two additional formal patterns, the **S**erver **R**eplicator (SR) and the ASV pattern, are introduced. In cases of high demand (e.g., a raising number of requests or a DDoS attack), the SR pattern replicates instances of a parametric server on demand while the ASV pattern represents a modularized specification of the ASV protocol, which provides a defense mechanism against DDoS attacks for a parametric client-server request-response system. Under DDoS attacks, the goal is to provide stable availability, i.e., that with very high probability service quality remains very close to a threshold, regardless of how bad the DDoS attack can get. Quantitative analysis of the two patterns has shown that the ASV pattern does not provide stable availability and that the SR pattern cannot provide stable availability at a reasonable cost. However, for the composition of ASV and SR, ASV^+SR (see Fig. 2), it has been shown that stable availability at a reasonable cost can be achieved.

Based on this example, we propose a general approach for enhancing safety and security of distributed systems through formal patterns:

1. Develop executable formal models of distributed systems in rewriting logic, supported by the Maude system.
2. Make these specifications modular and adaptive using instances of formal patterns. A catalog thereby provides highly reusable formal patterns such as the Meta-Object, ASV, and SR patterns.
3. Formally analyze these models to verify qualitative and quantitative properties using the Maude tool environment (e.g., parallelized statistical model checking supported by PVeSTA is able to analyze large system models).
4. Identify reusable formal patterns in the model and add their formal specifications to the pattern catalog.

4 Research Plan for Future Work

The main goal of the proposed research is to contribute a formal pattern-based approach and framework for the design of correct-, secure-, and safe-by-construction distributed systems, aided by a rich tool environment. We propose three main areas of future research: (i) build a rich and comprehensive catalog of formal patterns, (ii) identify security, safety, and other properties that are preserved by pattern composition and proof their preservation, and (iii) improve the existing tool support.

To build a rich and comprehensive catalog of formal patterns, existing patterns that are not yet explicitly modelled as a formal pattern need to be identified and formally specified.

In [6], it has been shown that the cookies protocol (a DDoS defense protocol), if wrapped around a system, preserves the safety properties of the wrapped system. We conjecture that the same is true for the ASV and $ASV^{+}SR$ protocols. In a first step, we want to prove that the ASV protocol also retains safety properties of the wrapped client-server request-response system. In the future, we want to identify such properties of other patterns and prove that they are preserved when the pattern is applied to a system. Having property preserving formal patterns improves their composability and reduces the formal verification effort as specific properties are, by construction, preserved in the composed model.

Furthermore, we propose to improve existing tool support in two main areas: (a) the robustness of existing tools and (b) code generation from executable formal models. Since we want to build systems in which many participants are communicating with each other and perform quantitative analyses on such systems, we need analysis and verification tools that scale with the size of these systems. In particular, analysis tools such as the PVeStA [3] statistical model checker, which drastically increases the scalability of statistical model checking through parallelization, need to be improved in terms of fault tolerance. Finally, to incorporate the proposed approach in an software engineering process, code generation techniques are needed. Thereby, based on correct-, secure-, and safe-by-construction specifications, correct-, secure-, and safe-by-construction implementations are generated.

5 Conclusion

In this paper we have presented the research goal of a formal pattern-based approach and framework for the design of correct-, secure-, and safe-by-construction distributed systems, aided by a rich tool environment. We gave a description of formal patterns, including the example of the Meta-Object pattern, and gave references to existing work that shows that formal patterns can help deal with security and safety issues and that formal analysis can help evaluate patterns in various contexts. In particular, we gave a description of the general formal pattern-based approach and concluded this paper with a summary of a research plan for future work.

References

1. G. Agha, S. Frolund, R. Panwar, and D. Sturman. A Linguistic Framework for Dynamic Composition of Dependability Protocols. *IEEE ICPADS*, 1:3–14, 1993.
2. G. Agha, J. Meseguer, and K. Sen. PMaude: Rewrite-based specification language for probabilistic object systems. *ENTCS*, 153(2):213–239, 2006.
3. M. AlTurki and J. Meseguer. PVeStA: A parallel statistical model checking and quantitative analysis tool. In *CALCO*, volume 6859 of *LNCS*, pages 386–392, 2011.
4. Arash Ferdowsi. Yesterday's Authentication Bug. `http://blog.dropbox.com/?p=821` (01/2012).
5. M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical report, University of California at Berkeley, 2009.
6. R. Chadha, C. A. Gunter, J. Meseguer, R. Shankesi, and M. Viswanathan. Modular Preservation of Safety Properties by Cookie-Based DoS-Protection Wrappers. In *FMOODS*, volume 5051 of *LNCS*, pages 39–58, 2008.
7. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *All About Maude - A High-Performance Logical Framework: How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *LNCS*. Springer, 2007.
8. J. Eckhardt, T. Mühlbauer, M. AlTurki, J. Meseguer, and M. Wirsing. Stable Availability under Denial of Service Attacks through Formal Patterns. In *FASE*, volume 7212 of *LNCS*, 2012.
9. T. Erl. *SOA Design Patterns*. Prentice Hall, 2008.
10. E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.
11. S. Khanna, S. Venkatesh, O. Fatemieh, F. Khan, and C. Gunter. Adaptive Selective Verification. In *IEEE INFOCOM*, pages 529–537, 2008.
12. M. Wirsing et al. Sensoria Patterns: Augmenting Service Engineering. In *ISoLA*, volume 17 of *CCIS*, pages 170–190, 2008.
13. J. Meseguer. Conditional Rewriting Logic as a Unified Model of Concurrency. *TCS*, 96(1):73–155, 1992.
14. J. Meseguer and C. Talcott. Semantic models for distributed object reflection. In *ECOOP*, volume 2374, pages 1–36. LNCS, 2002.
15. P. C. Ölveczky and J. Meseguer. Semantics and pragmatics of Real-Time Maude. *HOSC*, 20(1–2):161–196, 2007.
16. M. Schumacher, E. Fernandez-Buglioni, D. Hybertson, F. Buschmann, and P. Sommerlad. *Security Patterns*. Wiley, 2005.