Autonomy Features and Feature Composition in REBECA

Helge Parzyjegla[1], Arnd Schröter[2], Daniel Graff[2], Anselm Busse[2],
Alexej Schepeljanski[3], Jan Richling[2], Matthias Werner[3], and Gero Mühl[1]

[1] Architecture of Application Systems, University of Rostock, Germany
[2] Communication and Operating Systems Group, Berlin Institute of Technology, Germany
[3] Operating Systems Group, Chemnitz University of Technology, Germany

ABSTRACT

This paper explains how autonomy features targeting self-healing and self-optimization can be implemented and composed into REBECA, a publish/subscribe middleware which has been designed for distributed event-driven applications.

1. INTRODUCTION

Publish/subscribe is a flexible communication style well suited for event-driven applications. Application components interact by *publishing* notifications about occurred events and by *subscribing* to notifications of those events they are interested in. As publishers do not necessarily need to know their subscribers and vice versa, components are only loosely coupled and enable applications to be easily adaptable, flexibly extendable, and to run efficiently even in highly dynamic environments. The latter, however, requires that the underlying publish/subscribe infrastructure responsible to deliver a published notification to all its subscribers is also able to cope with frequently changing network conditions and application behavior. Since in these environments a manual administration is expensive at best, if not infeasible, modern publish/subscribe middleware implementations need to be context- as well as self-aware to autonomously organize and adapt themselves.

Therefore, our publish/subscribe middleware REBECA (`http://rebeca-middleware.org`) has been equipped with autonomy features. REBECA brokers exhibit strategies to self-organize their overlay network, to adapt and optimize the employed publish/subscribe routing algorithm used to exchange subscriptions and notifications, and to recover from failing nodes without loss of messages. Thereby, each feature is implemented by a dedicated algorithm encapsulated in a pluggable module. Considered in isolation, each algorithm works perfectly, but when running in parallel, autonomy features usually interfere and raise severe difficulties. Thus, composing single autonomy features to comprehensive solutions is challenging. It often leads to unexpected side-effects, unanticipated behavior and, for all non-trivial cases, it rarely succeeds on the first attempt.

Following, in Sect. 2, we introduce REBECA's flexible architecture, which is the foundation for Sect. 3, where we present challenges and approaches for the composition of autonomy features. We conclude with an outlook in Sect. 4.

ICAC'11, June 14–18, 2011, Karlsruhe, Germany.
ACM 978-1-4503-0607-2/11/06.

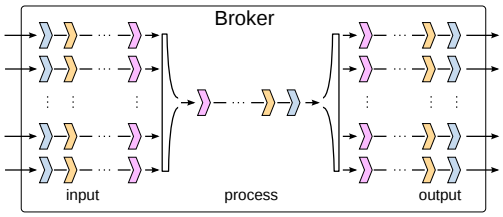

Figure 1: Pipeline architecture of a broker

2. REBECA ARCHITECTURE

REBECA's architecture [2] is centered around functional modularity. Therefore, all functional aspects of a REBECA broker and even those that are usually considered mandatory are seen as subject to feature composition to provide a maximum degree of configuration freedom. Thus, a broker is basically just a container for a selected set of pluggable components which, in turn, implement the broker's functionality provided to clients and applications. Following this idea, a broker only supports the concept of *processing stages* and *message channels* into which actual feature components are plugged as determined by the broker's configuration.

As shown in Fig. 1, a broker has only three processing stages that every message has to pass: the *input stage* after receiving, the *main processing stage*, and the *output stage* before sending. Additionally, the stages have different contexts and scopes. In the input and output stage, there is a dedicated message channel for each incoming and outgoing connection to a neighboring broker that can be used, for example, for message serialization and encryption. The main processing stage, however, has only one global channel for all messages that is suited, for example, to make routing decisions and determine to which neighbor brokers the message needs to be forwarded. Plugged feature components may interfere and manipulate messages in all stages to realize a certain functional aspect. For this, plugins may modify, transform, delete messages or even insert new ones.

In REBECA, even mandatory publish/subscribe functionality is implemented in terms of plugins. Thereby, a broker's functional core becomes easily exchangeable and replaceable with custom implementations that may better suit one's needs. Plugins are available for different publish/subscribe routing algorithms, for serialization and encryption, for different transport protocols and network simulator bindings as well as for autonomy maintenance and optimization of the broker's overlay network and routing algorithm.

3. AUTONOMY FEATURES

Quality of Service (QoS) and *fault tolerance* are important aspects of publish/subscribe systems. To avoid inefficient and invalid configurations in dynamic environments, the system must provide means to autonomously organize, optimize, and heal itself.

Self-organizing Topologies (SOT). Although targeting dynamic environments, publish/subscribe overlay topologies have often been assumed to be static. However, in settings where message patterns and network conditions are subject to frequent changes, static topologies inevitably lead to suboptimal system performance. Therefore, REBECA employs a self-organizing algorithm that autonomously adapts the structure of the overlay network [1]. The algorithm is based on an on-line heuristic which considers the characteristics of links, the performance of brokers as well as patterns occurred in the network's message flows. The algorithm first identifies inefficient links and potential replacement candidates. Thereafter, the impact of a possible reconfiguration is estimated and, if beneficial for all participating nodes, the topology gets reconfigured while ensuring message ordering and avoiding message losses.

Self-optimizing Routing (SOR). Traditionally, all brokers of a publish/subscribe network use the same routing strategy for exchanging messages which is usually determined at design time or set at system start-up. To better adapt the network to varying message flows, we introduced a self-optimizing publish/subscribe routing scheme that is based on *hybrid routing algorithms*. Hybrid algorithms enable a fine-grained, edge-wise routing configuration. Therefore, for each link, always the most beneficial routing strategy can be used depending on the relation of published notifications and forwarded subscriptions which may dynamically change over time. Based on a local decision criterion, neighboring brokers can, thus, renegotiate the applied strategy on their shared link using a simple coordination protocol [4].

Fault-tolerant Topologies (FTT). Starting from a certain size, scalable networks must consider faulty nodes and links. Within an acyclic publish/subscribe overlay network, such a fault almost always separates the network and disrupts the message flow from publishers to subscribers. To countervail message loss, the following steps are taken. First, the topology is repaired by substituting all links to a failed broker by new connections to an appropriate replacement broker. Thereafter, all affected neighboring brokers synchronize their filter tables to reestablish a valid routing configuration. Finally, messages lost in transit on the failed link or node are replayed from local caches of neighboring brokers.

Feature Composition. So far, we considered each autonomy algorithm in isolation. Hence, the challenge is to compose these in a way that preserves their individual properties. More precisely, the composition of the optimizing algorithms SOT and SOR should result in a better system performance than any of the two can achieve alone while both should not be disturbed by the fault-tolerance algorithm FTT. Beneficially, each algorithm was encapsulated as plugin and integrated into REBECA's pipeline architecture. As pipeline plugins just process or manipulate messages, they are not directly coupled by any other interface. Nevertheless, indirect dependencies may exist that need to be identified and considered before composition.

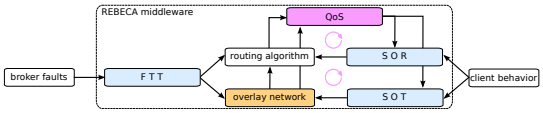

Figure 2: Dependencies

We systematically analyzed the plugins, the encapsulated algorithms, and their dependencies from and impact on system components on different system layers. The resulting dependency graph is shown in Fig. 2 highlighting critical points of the composition. FTT and SOT, for example, reconfigure or adapt the overlay network which, if not synchronized, may lead to inconsistent topologies. Furthermore, SOT and SOR are part of a control loop both affecting the same QoS measure which may lead to unintended oscillations. Identifying problematic interactions, their root causes, and involved components alleviates the design of effective counter measures significantly. In fact, based on this analysis, we derived a transaction scheme for SOT which makes the algorithm resistant to faults that occur during reconfiguration process while enabling the FTT algorithm to repair arbitrary topologies. Furthermore, we introduced delayed unsubscriptions for SOR that reduce the algorithm's reactivity and, thus, efficiently damp oscillations. In this context, REBECA's pipeline architecture primarily helped us to minimize the direct inter-feature communication via additional interfaces that became necessary.

4. CONCLUSION

As shown in this paper, we were able to compose different autonomy algorithms within our middleware. This was possible due to the fact that the middleware itself was developed with extensibility in mind. However, doing so required adaptation of the individual algorithms. Therefore, our future research targets towards true composability in the sense of [3]. This requires the development of an architectural framework together with design rules for such algorithms. This way, each algorithm obeying these rules can simply be integrated while composability is given by construction.

5. REFERENCES

[1] M. A. Jaeger, H. Parzyjegla, G. Mühl, and K. Herrmann. Self-organizing broker topologies for publish/subscribe systems. In L. M. Liebrock, editor, *The 22nd Annual ACM Symposium on Applied Computing (SAC'07)*, pages 543–550, Seoul, Korea, Mar. 2007. ACM.

[2] H. Parzyjegla, D. Graff, A. Schröter, J. Richling, and G. Mühl. *Design and Implementation of the Rebeca Publish/Subscribe Middleware*, pages 124–140. Springer Verlag, Sept. 2010.

[3] J. Richling. *Komponierbarkeit eingebetteter Echtzeitsysteme*. PhD thesis, Feb. 2006.

[4] A. Schröter, D. Graff, G. Mühl, J. Richling, and H. Parzyjegla. Self-optimizing hybrid routing in publish/subscribe systems. In *Proceedings of the 20th International Workshop on Distributed Systems: Operations and Management, DSOM 2009*, volume 5841 of *Lecture Notes in Computer Science*, pages 111–122. Springer, 2009.