Interfaces and Concepts to Build Large Resilient and Predictable Systems (Extended Abstract)

Raimund Kirner, Olga Tveretina
School of Computer Science
University of Hertfordshire
Hatfield, United Kingdom
{r.kirner, o.tveretina}@herts.ac.uk

Progress in technology not only results in tools of improved capabilities, but also pushes towards integration to create larger systems with unprecedented capabilities. Building such systems with safety-relevant services demands appropriate system interfaces and algorithmic concepts. In this paper we list examples of ingredients to build large resilient and predictable systems.

keywords: resilience; predictability; large systems; system of systems; composability

1 Introduction

Building large systems is always a trade-off between the requirements and the available resources. One the rather large scale system examples is the internet, with more than one Billion websites. The internet has proven to be relatively resilient to different types of system faults or attacks. However, C.Hall et al. have analysed the resilience of the internet interconnection ecosystem in more detail, and concluded that the dependability of the internet has its limits [1]:

"The economics do not favour high dependability of the system as a whole as there is no incentive for anyone to provide the extra capacity that would be needed to deal with large-scale failures."

While this is a valid design choice for the classical applications of the internet based on information exchange and data retrieval, there are an increasing number of large systems that would demand higher resilience than that provided by the internet interconnection ecosystem. For example, remote surgery would need communication lines with high availability between the hospital and the remote surgeon. Another example are smart cities, where we just stand at the beginning to envision what services are possible to make peoples' life more convenient and safe. In particular, safety-critical services are also envisioned, like the health monitoring and alerting for elderly people. To build such large systems with sufficient degree of resilience and predictability, we need to use adequate system interfaces and concepts. In this paper we provide some examples of what that can be.

2 Design Patterns for Large and Resilient Predictable Systems

In the following we introduce examples for systems design patterns that help to make systems for resilient and predictable.

2.1 Nearly Autonomous Systems

To design large systems, it is important to subdivide the system into subsystems, i.e., building a system of systems. To support resilience and predictability of critical services, it is important to design the corresponding subsystems providing that services as a *nearly autonomous system*. A nearly autonomous system is a subsystem with external communication that is able to provide its service even in the event of failure on the external communication channels. At the same time, it is recommended to use interfaces and algorithms that provide resilience against faulty data received on the external communication interface.

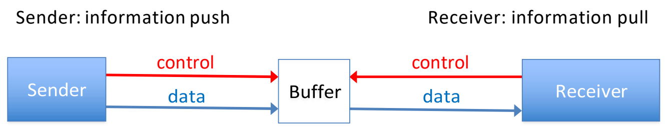


Figure 1. Push-Pull Communication Interface

An example for a resilient interface is the data exchange via a shared buffer, with decoupled write and read access. The sender controls when to write the data via an information push and the receiver controls when to read the data via an information pull [4]. This push-pull interface provides a temporal isolation between two subsystems. Such a communication interface supports composability as well as compositionality [7].

2.2 Interfaces for Mixed-Criticality Systems

Large systems tend to provide services of different criticality, where services of higher criticality have a higher weight for the overall system utility. To design such systems in a resilient and predictable way, it is also important to consider the message types and the type of message propagation [5]. For example, information exchange can be via event messages, state messages, or semi-state messages. State messages are preferred as they help to provide an immediate return to a consistent system state in case of erroneous failed communication. Communication channels of the system model have to be mapped to the physical communication medium. To provide fairness to the individual communication channels, approaches like bounded or time-triggered interfaces.

2.3 Lock-free Communication via the RNBC Protocol

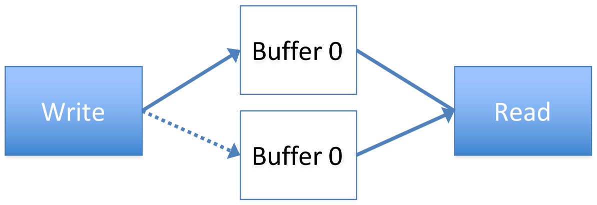
Realising a push-pull communication as described in Section 2.1, it is also important to realise the access to the shared communication buffer with temporal decoupling. A way to achieve this is the *Rate-bounded Non-Blocking Communication* (RNBC) protocol [6]. The core principle of RNBC is to have lock-free communication with one writer and an arbitrary number of readers. RNBC is rather simple, with the implementation shown in Figure 2.a. However, the important property of RNBC is to have a formal schedulability criterion that ensures correct communication. As shown in Figure 2.b, RNBC uses a double buffer to ensure the reading of consistent data. With c_r, c_w being the worst-case execution time of the reader and writer code, and *mint* being the minimum inter-arrival time between two messages, the following schedulability criterion guarantees lock-free and consistent communication via RNBC:

$$c_w + c_r \leq mint$$

```
int buff[2];           // shared msg buffer
int wbuff = 0;         // buffer index of write

void rnbc write msg(int msg) {
   buff[wbuff] = msg;
   wbuff = 1−wbuff; // swap read / write
      buffer
}

int rnbc read msg() {
   int rbuff = 1−wbuff;
   return buff[rbuff];
}
```



a) RNBC implementation b) RNBC double buffering

Figure 2. RNBC: Rate-bounded Non-Blocking Communication Protocol

2.4 Utility-based Service Optimisation

To design systems in a resilient way, we model the utility of individual services. Instead of using single design requirement limits like maximum delay or minimum throughput, we model these parameters via a utility function [3]. Figure 3 shows an example for modelling the utility of the throughput of a service. Using these utility values of the individual services, we can optimise the overall system utility in case of a situation that causes a resource shortage [2].

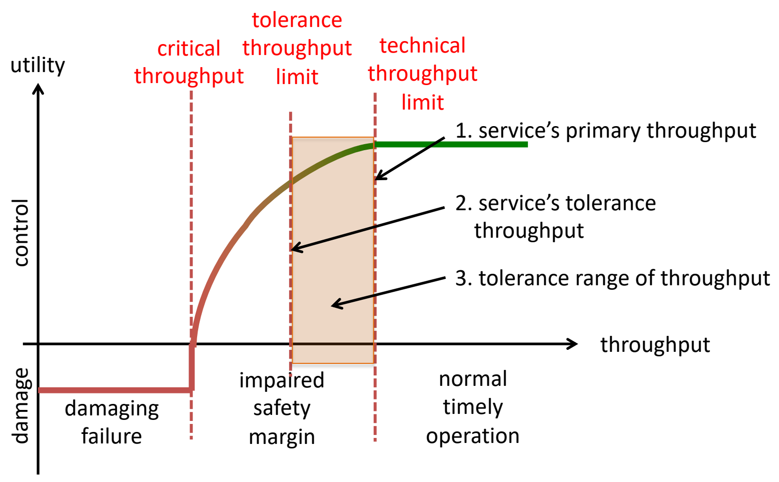

tolerance throughput limit
critical throughput
technical throughput limit
utility
control
damage
1. service's primary throughput
2. service's tolerance throughput
3. tolerance range of throughput
throughput
damaging failure
impaired safety margin
normal timely operation

Figure 3. Service Utility Optimisation based on Throughput

3 Summary and Conclusion

In this research, we have made the case towards adequate system interfaces and concepts to achieve resilient and predictable large systems. We also listed a few examples of such system interfaces and concepts.

References

[1] C. Hall, R. Anderson, R. Claytona, E. Ouzounis, and P. Trimintzios. Resilience of the internet interconnection ecosystem. Summary report of ENISA study, European Network and Information Security Agency (ENISA), Apr. 2011. Available online at https://www.enisa.europa.eu/publications/interx-report.

[2] S. Iacovelli, R. Kirner, and C. Menon. ATMP: An adaptive tolerance-based mixed-criticality protocol for multi-core systems. In *Proc. 13th International Symposium on Industrial Embedded Systems (SIES'18)*, Graz, Austria, June 2018.

[3] R. Kirner. A uniform model for tolerance-based real-time computing. In *Proc. 17th IEEE Int'l Symposium on Object/Component/Service-oriented Real-Time Distributed Computing*, pages 9–16, Reno, Nevada, USA, June 2014.

[4] P. Puschner and B. Frömel. Composable component interfaces for time-triggered systems. *In Proc. 8th Mediterranean Conference on Embedded Computing (MECO'19)*, Jun. 2019.

[5] S. Maurer and R. Kirner. Cross-criticality interfaces for cyber-physical systems. In *Proc. 1st IEEE Int'l Conference on Event-based Control, Communication, and Signal Processing*, Krakow, Poland, June 2015.

[6] P. Puschner and R. Kirner. Interfacing to time-triggered communication systems. In *Proc. 22nd IEEE Int'l Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing*, May 2019.

[7] P. Puschner, R. Kirner, and R. Pettit. Towards composable timing for real-time software. In *Proc. 1st International Workshop on Software Technologies for Future Dependable Distributed Systems*, Tokyo, Japan, Mar. 2009.