Flec: a versatile programming framework for eventually consistent systems

Jim Bauwens
Software Languages Lab
Vrije Universiteit Brussel
Belgium
jim.bauwens@vub.be

Elisa Gonzalez Boix
Software Languages Lab
Vrije Universiteit Brussel
Belgium
egonzale@vub.be

Abstract

Modern distributed applications increasingly replicate data in order to guarantee both high availability of systems and an optimal user experience. Conflict-Free Replicated DataTypes (CRDTs) are a family of data types specially designed for highly available systems which guarantee some form of eventual consistency. However, currently CRDT implementations are hard to integrate with existing applications and/or programming languages. In this extended abstract we describe Flec, a versatile programming framework for operation-based CRDTs that ultimately can be run in any environment supporting WebAssembly.

CCS Concepts: • **Software and its engineering → Consistency**; **Synchronization**; **Middleware**; **Reflective middleware**; • **Computer systems organization → Distributed architectures**.

Keywords: Replication, CRDTs, Middleware, Reflection, WebAssembly, Eventual consistency

ACM Reference Format:
Jim Bauwens and Elisa Gonzalez Boix. 2020. Flec: a versatile programming framework for eventually consistent systems. In *7th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC '20), April 27, 2020, Heraklion, Greece.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3380787.3393685

1 Introduction

Many modern distributed systems keep multiple copies of data (replicas) between distributed components. When a partial failure occurs, the copies ensure availability of the data in the system. This also improves performance by lowering request latencies and as a result, provides a better user experience as requests are served faster. A system is expected to provide users with up-to-date information, but keeping replicas consistent is a complex task. One of the main reasons to the complexity of ensuring consistent behaviour is that there is no notion of a global clock in distributed systems. This has as result that the order of updates applied to different replicas in the system cannot be precisely determined, which complicates determining when updates are concurrent and how conflicts caused by concurrent updates should be resolved.

Conflict-Free Replicated Data Types [11] (CRDTs) are promising data structures for eventually consistent systems as programmers do not need to manually deal with conflicts. CRDTs are replicated data structures which can be concurrently updated without requiring synchronisation among replicas. To this end, CRDTs constrain the type of operations which can be applied on them. Assuming no new updates happen to a set of replicas, they will eventually converge to the same state without conflicts.

A lot of CRDT research has focused on providing formal specifications of different data types (e.g. OR-Sets, replicated growable arrays, embeddable counters and more) [1, 4, 7, 11, 12], but limited work has focused on embedding CRDT in actual language implementations [9, 10].

Developers using existing libraries need to handle many distribution aspects themselves, such as deciding on how to handle discovery of new network acquaintances and in what way that they will cope with a dynamically changing system [3]. This greatly raises the barrier for utilising CRDTs in applications.

In this paper we introduce Flec, a modular programming middleware that enables the development and use of CRDTs in a flexible manner.

Firstly, Flec provides a flexible networking framework that allows programs to work on different platforms. Moreover, Flec exposes CRDT internals by means of a Meta Object Protocol (MOP)[8], easing development of new variants through reflection. The MOP allows developers to hook into several points of the CRDT replication process. Finally, it targets WebAssembly[1], a powerful assembly language that is designed to be a portable target that can run on a multitude of

PaPoC '20, April 27, 2020, Heraklion, Greece

ACM ISBN 978-1-4503-7524-5/20/04.
https://doi.org/10.1145/3380787.3393685

[1] https://webassembly.org/

platforms. This makes Flec ideal for experimenting with and developing new replicated data structures.

2 An overview of Flec

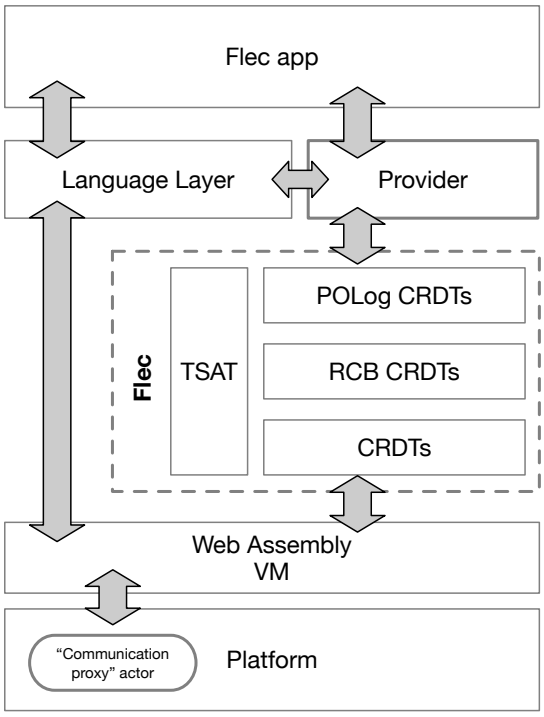

Figure 1. Flec architectural overview

Flec is implemented in AssemblyScript[2], a strict subset of TypeScript that can be compiled to WebAssembly. TypeScript is a typed superset of JavaScript. The goal is to have any language that can target WebAssembly to be able to utilise the framework, by means of small language providers that provide APIs to Flec for different languages.

Figure 1 gives an overview of the architecture of an application that uses CRDTs provided by the Flec framework. Applications are compiled from high-level languages and can interact with Flec by using the language and language provider layers. Language providers ensure that values, data and code structures native to a language can be mapped to WebAssembly constructs, allowing them to be stored in or interact with Flec CRDTs.

The main component of Flec is TSAT, which incorporates the concepts of ambient-oriented programming [5, 6], a paradigm geared towards distributed mobile computing. In ambient-oriented programming developers are provided with an actor-based programming model where actors can communicate and coordinate over dynamic networks. To this end TSAT brings support for ambient acquaintance management, non-blocking message passing between actors, failure handling through message buffering, leasing and future-like synchronisation constructs.

Depending on what platform TSAT is running on, the exact means of transportation can be different: in web browsers it could be using WebSockets or WebRTC, on embedded devices (such as the ESP32, a lightweight, power-efficient integrated system-on-a-chip platform) it could be over pure TCP sockets. For this it is engineered to be network agnostic and only knows about sending messages between actors. The idea is that a special router actor (the *'communication proxy'* actor in the diagram) has to be implemented in the host platform and made available to the WebAssembly environment. TSAT will use this router actor for forwarding messages to Actors over a network when needed.

2.1 Using CRDTs in Flec

We currently have a basic provider for Lua[3], a highly embeddable programming language, allowing the use of simple CRDTs in the language. No explicit language provider is required for applications written AssemblyScript however, as it is implicitly provided by TSAT which is written in AssemblyScript. Because TSAT operates under the language layer, it is actually possible to use the same data-structures and have interaction with and from other languages.

Listing 1 and 2 show the use of a counter CRDT in AssemblyScript and Lua respectively.

```
let counter = new CounterCRDT('shared_counter');

counter.setUpdateHandler(value => {
  console.log('Counter updated', value);
});

counter.increment(1);
```

Listing 1. Using a counter CRDT in AssemblyScript

```
local counter = CounterCRDT("shared_counter")

counter:setUpdateHandler(function (value)
      print('Counter updated', value)
end))

counter:increment(5)
```

Listing 2. Using a counter CRDT from Lua

In both the AssemblyScript and Lua versions, the `CounterCRDT` constructor takes a string representing a nominal type used for other actors (locally or on other network nodes) to discover this CRDT. It then creates a counter CRDT instance which can be discovered in the network by means of the `shared_counter` string. `setUpdateHandler` is used to set a callback that will be invoked when the CRDT is updated. Mutation of the CRDT happens by calling the `increment` or `decrement` operations on the CRDT reference. Flec will ensure that these operations are replicated across all devices hosting a CRDT replica.

Another example of CRDT is an AWSet as seen in listing 3, which replicates a set of items using add-wins semantics.

[2]https://github.com/AssemblyScript/assemblyscript

[3]https://www.lua.org/

```
let set : AWSet = AWSet('shared_set');

set.setUpdateHandler(set => {
  console.log(set.join(', '));
});

set.add("element");
set.add("this is another item");

set.remove("element")

if (set.lookup("element"))
  console.log("Element is in the set");
else
  console.log("Element is not in the set");
```

Listing 3. Using an AWSet CRDT in AssemblyScript

Just like `CounterCRDT`, `AWSet` takes a string representing a nominal type for linking replicas together, and with `setUpdateHandler` a callback function can be set which will be applied when the set is updated. Mutation of the set happens by calling the `add` or `remove` operations on the set reference. Using the `lookup` method items can be tested if they exist in the set. The `toList` method returns an array containing all elements as a non-replicated list.

2.2 Defining new CRDTs in Flec

Right now Flec only allows for CRDT definition (i.e. adding custom behaviour) from within AssemblyScript, but with extended language providers this should eventually become possible from within other languages.

To implement a new CRDT, there are several classes provided by Flec that provide some base functionality, listed below.

- `CRDT`: General-purpose CRDT class that provides constructs for operation- and state-based CRDTs.
- `CRDT_RCB`: Extended CRDT class for operation-based CRDTs that implements reliable causal broadcasting (RCB) for causal operation ordering. To this end, every operation is tagged with a vector-clock, which receiving nodes will use for determining causal relations.
- `POLog`: A CRDT class that allows the creation of pure-operation (POLog) based CRDTs [2]. In a POLog-based CRDT every operation is stored in a log, and the state of a replica is determined by performing computations on this log.

Any CRDT type implemented using these classes will be able to fully communicate over the network with other replicas, inheriting discovery and communication from TSAT. Due to space constraints, Appendix A shows the implementation of an AWSet using the POLog class.

3 Conclusion

Conflict-free Replicated Data Types (CRDTs) are a promising programming abstraction to replicate data in a distributed system as they guarantee that eventually all replicas will end up in the same state. In this paper we introduce Flec, a highly versatile framework that aims to provide CRDTs constructs to a multitude of programming languages. It provides developers with a flexible environment to define, implement and use CRDTs. By targeting WebAssembly, the framework can run on many different platforms, and allows us to experiment with CRDTs in various settings.

References

[1] C. Baquero, Paulo S. Almeida, and C. Lerche. 2016. The Problem with Embedded CRDT Counters and a Solution. In *Proceedings of the 2Nd Workshop on the Principles and Practice of Consistency for Distributed Data (PaPoC '16)*. ACM, New York, NY, USA, Article 10, 3 pages. https://doi.org/10.1145/2911151.2911159

[2] C. Baquero, P. S. Almeida, and A. Shoker. 2017. Pure Operation-Based Replicated Data Types. *CoRR* abs/1710.04469 (2017). arXiv:1710.04469

[3] Jim Bauwens and Elisa Gonzalez Boix. 2019. Memory Efficient CRDTs in Dynamic Environments. In *Proceedings of the 11th ACM SIGPLAN International Workshop on Virtual Machines and Intermediate Languages (VMIL 2019)*. Association for Computing Machinery, New York, NY, USA, 48–57. https://doi.org/10.1145/3358504.3361231

[4] S. Burckhardt, A. Gotsman, H. Yang, and M. Zawirski. 2014. Replicated Data Types: Specification, Verification, Optimality. In *Proceedings of the 41st ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '14)*. ACM, New York, NY, USA, 271–284. https://doi.org/10.1145/2535838.2535848

[5] T. Van Cutsem, S. Mostinckx, E. Gonzalez Boix., J. Dedecker, and W. De Meuter. 2007. AmbientTalk: Object-oriented Event-driven Programming in Mobile Ad hoc Networks. In *XXVI International Conference of the Chilean Society of Computer Science (SCCC'07)*. Iquique, Chile, 3–12. https://doi.org/10.1109/SCCC.2007.12

[6] J. Dedecker, T. Van Cutsem, S. Mostinckx, T. D'Hondt, and W. De Meuter. 2006. Ambient-Oriented Programming in AmbientTalk. In *ECOOP 2006 – Object-Oriented Programming*, Dave Thomas (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 230–254.

[7] R. Hyun-Gul, J. Myeongjae, K. Jin-Soo, and L. Joonwon. 2011. Replicated abstract data types: Building blocks for collaborative applications. *J. Parallel and Distrib. Comput.* 71, 3 (2011), 354 – 368.

[8] Gregor Kiczales, Jim des Rivières, and Daniel G. Bobrow. 1991. *The Art of Metaobject Protocol*. MIT Press, Cambridge, MA, USA.

[9] M. Kleppmann and A. R. Beresford. 2017. A Conflict-Free Replicated JSON Datatype. *IEEE Transactions on Parallel & Distributed Systems* 28, 10 (oct 2017), 2733–2746. https://doi.org/10.1109/TPDS.2017.2697382

[10] C. Meiklejohn and P. Van Roy. 2015. Lasp: A Language for Distributed, Coordination-free Programming. In *Proceedings of the 17th International Symposium on Principles and Practice of Declarative Programming (PPDP '15)*. ACM, New York, NY, USA, 184–195. https://doi.org/10.1145/2790449.2790525

[11] M. Shapiro, N Preguiça, C. Baquero, and M. Zawirski. 2011. *A comprehensive study of Convergent and Commutative Replicated Data Types*. Technical Report 7506. INRIA.

[12] P. Zeller, A. Bieniusa, and A. Poetzsch-Heffter. 2014. Formal Specification and Verification of CRDTs. In *Formal Techniques for Distributed Objects, Components, and Systems*, E. Ábrahám and C. Palamidessi (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 33–48.

Appendices

A AWSet POLog definition

Listing 4 shows the implementations of an AWSet in AssemblyScript. It utilises the `POLog` class which provides a

pure-operation based back-end. The implementation can almost be directly be mapped to the original POLog AWSet specification[2]. Flec will handle all distribution and replication aspects.

```
class AWSet extends POLog {

    constructor(tag, callback) {
        super(tag);

        this.callback = callback;
    }

    isRedundantByOperation(e1, e2, isRedundant) {
        return e1.clock.precedes(e2.clock) && ( e2.operation
              == "clear" || e1.args[1] == e2.args[2] );
    }

    isRedundantByLog(entry) {
        let op = entry.operation;

        return op == "rmv" || op == "clear";
    }

    toList() {
        let state = this.getState();
        let set   = [];
        let list  = [];

        state.log.forEach(element => {
            let i = element.args[1];
            if (set[i] == null) {
                set[i] = element;
                list.push(i);
            }
        });

        set.forEach(entry => {
            let index = 0;
            if (entry.operation = "add") {
                list.push(index);
            }
        });

        return list;
    }

    add(element){
        this.performOp("add", [element]);
    }
    remove(element) {
        this.performOp("remove", [element]);
    }
    clear(element) {
        this.performOp("clear", [element]);
    }

}
```

Listing 4. AWSet definition