Towards A Unified Programming Model for Blockchain Smart Contract dApp Systems

Joshua Ellul
Centre for Distributed Ledger Technologies
University of Malta
Msida, Malta
joshua.ellul@um.edu.mt

Gordon J. Pace
Centre for Distributed Ledger Technologies
University of Malta
Msida, Malta
gordon.pace@um.edu.mt

***Abstract*—Developing smart contract decentralised application based systems typically involves writing code for various platforms, from the smart contract code residing on the underlying distributed ledger technology implementation to back end oracles and front end websites or mobile apps. In addition to the different technologies used for the different parts, the programmer is also burdened with implementing communication channels between the various parts. In this paper we propose a unified programming model allowing for developers to build such systems through a single code artifact, using a macroprogramming approach.**

***Keywords*-Blockchain, Smart Contracts, Programming**

I. Introduction

Smart contracts executing on top of a blockchain system allows for a decentralised trustless execution environment amongst involved parties. However, due to operational decentralisation, frequently the underlying instruction set or its execution model is constrained, with some platforms limiting computational expressivity e.g. Bitcoin script [3], and Findel [2] are not Turing-complete languages, whilst others limit the execution cost (in terms of time, space or both) e.g. the underlying Ethereum Virtual Machine [7] execution model requires payment of gas for execution resources. Due to these constraints, it is desirable, if not necessary, to delegate some of the execution off-chain. However, this comes with its challenges — when writing systems which straddle heterogeneous platforms, one requires (i) to identify which parts of the logic and data are to reside on which platform; (ii) to identify means of communication between the platforms, possibly using different modalities for different communication lines (e.g. triggering some logic on the other platform whose outcome is required to continue on the current platform may require a synchronous message and response to be sent, whilst asynchronous communication would suffice for logging information on the other platform); (iii) handling data conversion between the two platforms whenever data from one side is to be used by the other; and (iv) programming the different parts, possibly using different technologies. These requirements make the development of such systems challenging and error prone, and requires different expertise.

This scenario of requiring development of a single application working across heterogeneous platforms is not new or unique and similar challenges arose in other domains, most notably: (i) with the appearance of affordable programmable and reconfigurable hardware in the 80s and 90s, the development of systems which were to be partly deployed in software and partly in hardware gave rise to software-hardware codesign techniques e.g. [4]; (ii) wireless sensor networks are comprised of many resource constrained devices equipped with various sensors and actuators, giving rise to macroprogramming of systems in a manner which automatically deploys different parts of the code and data to different sensor nodes e.g. [1]. The standard approach in addressing this recurring challenge is that of designing a programming formalism to enable a monolithic source to be compiled seamlessly to different targets, and abstracting away communication between the platforms, including data conversion. Unless the different platforms are sufficiently different so as to enable automated partitioning, the only additional information a developer must include in a system to be split across different platforms, are annotations identifying where data and computation is to be located. This reduces complexity overheads when developing such systems and thus reducing potential for bugs and errors.

In this paper we propose a framework for a unified programming model for systems which allow for the specification of applications with data and computation which may be partially but not completely decentralised on a blockchain platform.

II. Related Work

In earlier work, we have looked at related aspects of the challenge. In [6], we looked at macroprogramming devices on the internet-of-things together with blockchain platforms. However, the work presented there was limited to triggering individual transactions based on the computation with little computation performed on the blockchain systems. We have extended this work in [5], in which we proposed a programming language, Porthos, intended to program smart contracts which span across different blockchain platforms. In contrast with the platform and language proposed in this paper, Porthos is not Turing complete, thus limiting the class of systems which can be programmed. Furthermore, in our previous work, we adopted an implicit partitioning approach to splitting a Porthos specification to different blockchain platforms. This was largely possible thanks to (i) tagging of

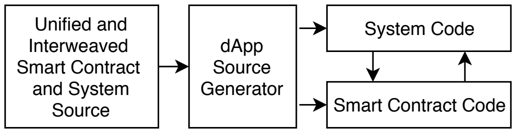

Fig. 1. Framework Overview

resources with the blockchain/DLT where they are stored; and (ii) the simpler processchain-like structure of Porthos programs. The framework we propose in this paper takes a more general approach, and also allows partitioning between on-chain and off-chain code at both data and control-flow level.

III. A Unified Programming Model for dApps

Decentralised application systems (dApps) are typically comprised of decentralised smart contract logic deployed on a blockchain with other centralised logic coded into a website or desktop system. dApps require developers to code the various system components separately which implicitly requires that the various components communicate with each other using well defined application programming interfaces (APIs). Such communication often involves specifying function signatures, including parameter and return types, which are often communicated using character strings – which results in the various compilers and development infrastructure being unable to ascertain whether such communication was correctly coded. A unified programming model for dApps is proposed herein.

Figure 1 depicts an overview of the framework. Source code for the decentralised smart contract and centralised components are coded within a single source base. The unified source is passed through the *dApp Source Generator* which produces the smart contract and system code to be deployed.

In the unified source the developer should provide annotations for code in respect to where various code fragments should be placed. Consider unified source for a simple property purchase dApp with functionality for purchasing of a property as follows:

```
@XOn(MainSystem) function buyProperty() {
  p = selectedProperty;
  @XOn(Chain) {
     Assert(p.isForSale);
     Assert(Tx.user.balance >= p.salePrice);
     p.owner = Tx.user;
     p.isForSale = false;
     Tx.user.balance -= p.salePrice;
  }
  displayPurchaseSuccess();
}
```

The `@XOn(MainSystem)` annotation defines that the function `buyProperty` should be generated to execute on the main system, whilst the internal code block annotated by `@XOn(Chain)` should execute in the decentralised smart contract — in which the main system will wait for the decentralised execution to complete before continuing execution. The smart contract code generated will almost be exactly as defined in the code above. However, to support seamless communication between the systems, the main system code generated would require changes as follows:

```
function buyProperty() {
  p = selectedProperty;
  XCall(Chain, "buyProperty", p);
  displayPurchaseSuccess();
}
```

The `XCall` function is a runtime library function that will support the cross-platform call to the `buyProperty` smart contract function (and will pass the parameter p).

Developers must consider not only locality of computation, but also locality of data. Therefore data fields can be marked up in a similar manner, and the code generator and runtime libraries support for automatic placement and communication of data values as required. Consider the following unified source:

```
@XAll() class User extends XDataObject {
  public XID id;
  @XOnlyOn(MainSystem) public String name;
  @XOnlyOn(Chain) public int balance;
}
```

The `XAll` annotation defines that the User class object will be stored on all platforms, whilst the use of the `XOnlyOn` annotation defines that the `name` field will only be available on the main system, whilst the `balance` field will only be available within the smart contract. As an example consider the generated smart contract code as follows:

```
class User {
  public GUID Chain_id;
  public int balance;
}
```

IV. Conclusions

In this paper we propose a unfied programming model for smart contract dApp based systems and demonstrate various annotations for computation and data locality definition. We are working on different implementations for various platforms to further demonstrate the usefulness of this framework.

References

[1] Asad Awan, Suresh Jagannathan, and Ananth Grama. Macroprogramming heterogeneous sensor networks using cosmos. In *ACM SIGOPS Operating Systems Review*, volume 41, pages 159–172. ACM, 2007.
[2] Alex Biryukov, Dmitry Khovratovich, and Sergei Tikhomirov. Findel: Secure derivative contracts for ethereum. In *International Conference on Financial Cryptography and Data Security*, pages 453–467. Springer, 2017.
[3] bitcoin.org.
[4] CAR Hoare and Ian Page. Hardware and software: The closing gap. In *Programming Languages and System Architectures*, pages 49–68. Springer, 1994.
[5] Adrian Mizzi, Joshua Ellul, and Gordon Pace. Technical report: Porthos macroprogramming blockchain systems. 2019.
[6] Adrian Mizzi, Joshua Ellul, and Gordon J Pace. Macroprogramming the blockchain of things. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1673–1678. IEEE, 2018.
[7] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.