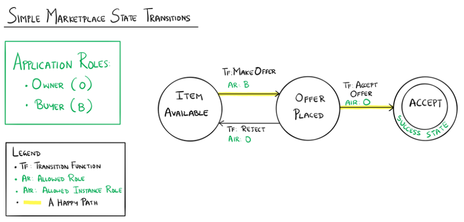
# TRAC: a tool for data-aware coordination
## (with an application to smart contracts)

João Afonso[1], Elvis Konjoh Selabi[2,3], Maurizio Murgia[3],
António Ravara[1], and Emilio Tuosto[3]

[1] NOVA School of Science and Technology
[2] Università di Camerino
[3] Gran Sasso Science Institute

We propose TRAC, a tool to support the coordination of distributed applications. The design of TRAC is inspired by the Azure initiative of Microsoft [4] which advocates the use of finite-state machines (FMSs) to specify the coordination of smart contract (SC for short). This idea is not formalised; in fact, Azure's FSMs are informal sketches aiming to capture the "correct" executions of SCs. For instance, the FSM for the simple market place (SMP) scenario borrowed from [5] (the textual description is ours):



The sketch declares the roles (Owner and Buyer) played by participants.
In the initial state `Item Available` the buyer is allowed to make an offer, moving the protocol to the `Offer Placed` state where two options are possible: the owner either accepts the offer (making the protocol reache the success state `Accept`) or rejects the offer (moving back the protocol to `Item Available`).
The labels of the transitions specify which role executes with operations to make the protocol progress.

The FSM informally specifies a protocol coordinating the participants enacting the roles owner and buyer, from a *global* standpoint; we call *coordination protocol* such specification. A coordination protocol can be regarded as *global view* –in the sense of choreographies [1,3]– where the state of the protocol determines which operations are enabled. This resembles the execution model of monitors [2]. In fact, as in monitors, coordination protocols encapsulate a state that –through an API– concurrent processes can have exclusive access to. The API is basically a set of operations guarded by conditions set to maintain an invariant on the encapsulated state (in the SMP scenario the operations are `MakeOffer`, `AcceptOffer`, and `Reject`). The key differences between coordination protocols and monitors [2] is that in the former $(i)$ participants are distributed and do not share memory, $(ii)$ the invocation of an operation whose guards is not valid in the current state is simply ignored without preempting the caller, and therefore $(iii)$ processes do not have to be awaken.

We aim to refine the approach of Azure so to enable algorithmic verification of relevant properties of data-aware coordination of protocols. In fact, as for monitors, the interplay among the operations that modify the state and the guards in the API can lead to unexpected behaviours when informal specifications are used. We illustrate this problem with some examples on the SMP example.

1. The sketch of SMP does not clarify if a participant can play more roles simultaneously; for instance, it is not clear if an owner must be a different instance than buyers.
2. The labels distinguish roles and instances (`AR` and `AIR`): in fact, it is assumed that there can be many instances of a same role. Scope and quantification of roles is not clear; for instance, a requirement specified in [5] reads "The transitions between the `Item Available` and the `Offer Placed` states can continue until the owner is satisfied with the offer made." This sentence does not clarify if, after a rejection, the new offer can be made by a new buyer or it must be the original one;
3. The sketch specify neither the conditions enabling operations in a given state nor how operations change the state of the contract's variables; should the price of the item remain unchanged when the owner invokes the `Reject`?

**Contributions** This paper proposes DAFSMs, a data-aware coordination model for orchestrated computation applicable to the description of multiparty protocols. The key novelties are: 1. the support for multiple participants, organised by roles, which can dynamically join a protocol; 2. the use of assertions to describe a protocol state and control how (parametrised) actions change it (in a style akin to Hoare triples); 3. a notion of well-formed models and a checking algorithm; 4. a tool for describing systems with DAFSMs, visualising them as FSMs, and checking their well-formedness. The applicability of TRAC will be demonstrated by showing how its features can specify and verify. Moreover, we will discuss the performances of the TRAC with an experimental evaluation. The source code of TRAC and our experimental data is available at https://github.com/loctet/TRAC.

## References

1. Object Management Group: Business Process Model and Notation, http://www.bpmn.org
2. Hansen, P.: Operating System Principles. Prentice-Hall (1973)
3. Kavantzas, N., Burdett, D., Ritzinger, G., Fletcher, T., Lafon, Y.: http://www.w3.org/TR/2004/WD-ws-cdl-10-20041217, working Draft 17 December 2004
4. Microsoft: The blockchain workbench. https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench (2019)
5. Microsoft: Simple marketplace sample application for azure blockchain workbench. https://github.com/Azure-Samples/blockchain/tree/master/blockchain-workbench/application-and-smart-contract-samples/simple-marketplace (2019)