interlace	**Interacting Decentralized Transactional and Ledger Architecture for Mutual Credit**

WP2

Iterative Architecture Requirements and Definition

Deliverable D2.3

Final Architecture

Horizon 2020	**Project funded by the European Commission** Information and Communication Technologies FET OPEN Launchpad Project Grant no. 754494

Contract Number: 754494

Project Acronym: INTERLACE

Deliverable No: D2.3

Due Date: 30/10/2018

Delivery Date: 31/01/2019

Author: Paolo Dini (UH), Giuseppe Littera, Luca Carboni (SARDEX), Eduard Hirsch (SUAS)

Partners contributed: Maria Luisa Mulas (SARDEX), Thomas Heistracher (SUAS), Aurelio Riccioli (Open Source Community)

Made available to: Public

Versioning

Version	Date	Name, organization
1	01/10/2018	Paolo Dini (UH)
2	18/11/2018	Paolo Dini (UH), Eduard Hirsch (SUAS), Luca Carboni (SARDEX)
3	07/01/2019	Paolo Dini (UH), Giuseppe Littera (SARDEX)
4	31/01/2019	Paolo Dini (UH)

Reviewers: Giuseppe Littera (SARDEX), Egon Börger (UNI PASSAU) and Aurelio Riccioli (Open Source Community)

Abstract

This report describes briefly the architecture that was implemented by the INTERLACE project, the current view of the whole Sardex application microservice architecture, and the architecture of the overall system as it scales to many countries and jurisdictions. The report concludes with some reflections on the role of a responsible and critical engineering approach towards the design and development of an inclusive, open, and shared infrastructure layer as the main legacy of the INTERLACE project.

Table of Contents

Chapter 1

Introduction

Paolo Dini and Giuseppe Littera

The final architecture of the INTERLACE platform can be understood to refer to three different systems. First, it is the architecture that was implemented as reported in deliverable D3.2 [5]; second, it is the architecture of the whole Sardex application, which includes the INTERLACE, blockchain-based transactional platform as one of its microservices; and, third, it is the long-term open shared architecture vision of the overall system as it scales to many countries and jurisdictions.

This report describes briefly all three, and concludes with some reflections on the role of a responsible and critical engineering approach towards the design and development of an inclusive, open, and shared infrastructure layer as the main legacy of the INTERLACE project.

Chapter 2

Final Architecture

Paolo Dini, Giuseppe Littera, Eduard Hirsch, Luca Carboni, and Aurelio Riccioli

The technical architecture of the system implemented is discussed in some detail in deliverable D3.2 [5]. Here we present just the high-level CTO model, which is somewhat analogous to a class diagram. We then present the high-level view of the microservice-based Sardex application architecture; and finally we discuss the long-term view of a global multi-layer open shared architecture within which the Sardex system becomes one of many possible applications.

2.1 CTO-Model of the System Implemented

The CTO-model is specific to the Hyperledger Composer framework and offers the possibility to set-up a ground model which drives the chaincode implementation. 'CTO' refers to the initial name of the language used to specify the model, 'Concerto'. Even though it was then changed to Go, the extension '.cto' remained. The primitive concepts of a .cto model are:

- Transactions
- Assets
- Participants
- Concepts
- Events
- Enum

As can be seen in Figure 2.1, the model created for the INTERLACE project has two main *transactions*, *CreditTransfer* and *DebitTransfer*, that inherit from *Transfer*. *DebitTansfer* actually also needs a second transaction named *DebitTransferAcknowledge* to be performed, thus it may be counted as a main transaction as well. Two other transactions supporting the functionality of the network are *InitBlockchain* and *CleanupPendingTransfer*.

Transfers create and update *assets*. When the main transfers are executed they change *Account* assets: namely, *SysAccount* and *MemberAccount* which inherit from *Account*. *DebitTransfer* also needs the *PendingTransfer* asset for creating transfers that have not been confirmed yet. *DeltaDebt* collects all transfers which caused a balance to go negative or that caused a negative balance to go more negative, to log when the debt has to be paid back. Please see the Appendix 3 for more details.

In order to perform credit or debit operations a *participant* needs to be registered. There are two participants derived from *Member*, called *Subscriber* and *Individual*. These members may own an account asset which can be used to transfer credits from one account to another.

On several occasions events are emitted which may be consumed by an application or a user to react to specific issues that have happened during a transaction. There are several different events that may be of interest to a user, but for the moment only three are implemented: *LimitViolation* warns of a possible account limit violation; *RequestDebitAcknowledge* asks a user to give an acknowledgement to a pending transaction; and *DebitAcknowledgeInvalid* informs the user that the acknowledgement for a pending transaction was denied or is invalid.

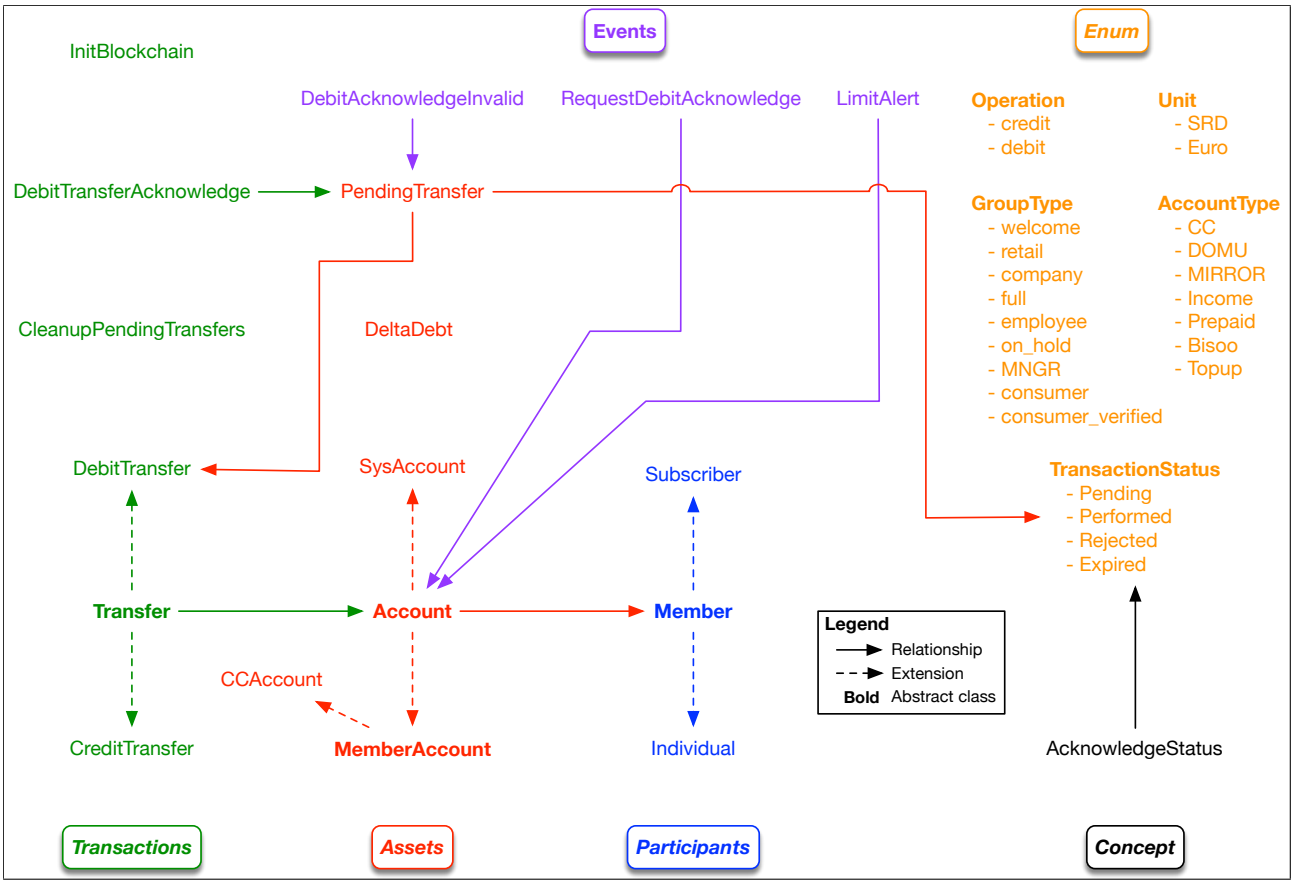

Fig. 2.1: **Hyperledger .cto model/class diagram of INTERLACE transactional platform**

Finally, there are a few *enum* types with states names for operations and units as well as account and group types. A detailed explanation of the model language can be found the official hyperledger composer documentation[1]

2.2 Sardex Application Architecture

Figure 2.2 provides a snapshot of the high-level application architecture of the Sardex system. The detailed Legend of the figure should make it possible to discern what the different components are and do. The heart of the system is the Permissioned Hyperledger Transaction Platform: this is the component that was implemented by INTERLACE at the level of a proof-of-concept prototype.

In this view, the different circuits operating in the different Italian regions are connected to the same network, and in fact to the same Hyperledger Channel. Thus, in this initial prototype the architecture is centralised, with the only possible outsourcing the Identity service that Auth0 or similar could perform.

Other infrastructural services, with red border, are shown as microservices: Search, Adverts, etc. The system also needs several kinds of user interfaces, shown with a blue border. The blockchain itself requires a GUI for the SysAdmin and one of the users, who may wish to inspect past transactions. Then, different kinds of users will need their own specialised GUIs: B2B end-user, B2C/B2E end-user, Broker, Partner (i.e. manager of a different circuit), general public, and SysAdmin. We have not shown a Regulator interface but that is likely to be implemented too.

Finally, the figure also shows possible future extensions of the system towards other types of blockchain. The two most notable and likely, at this point, are Ethereum, Stellar, and Holochain. Ethereum and

[1] https://hyperledger.github.io/composer/latest/reference/cto_language.html

Stellar could be used to extend the payment service towards inter-circuit trades. Stellar, in particular, is specialised for currency exchange. However, Ethereum may be easier to adopt since Hyperledger is developing an Ethereum interface.

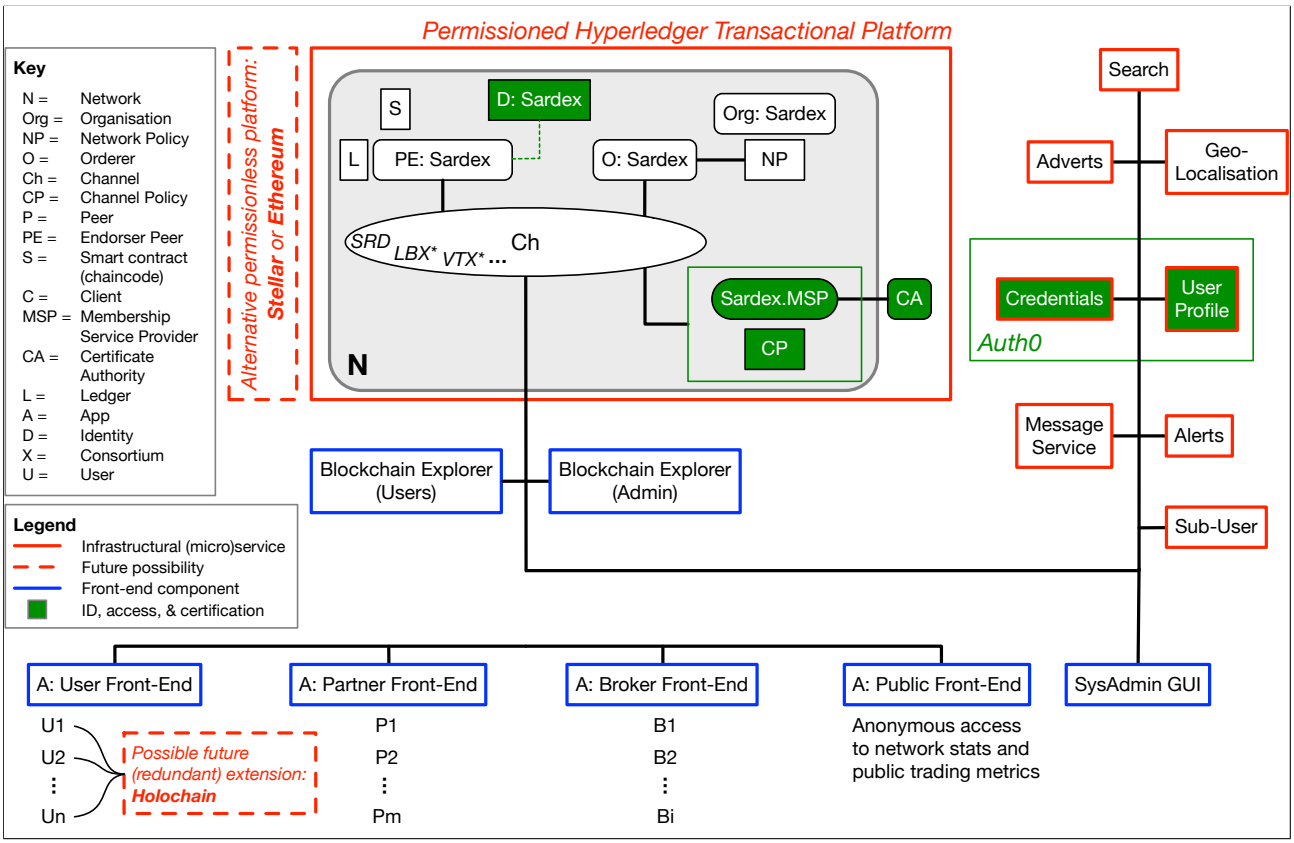

Fig. 2.2: **High-level Hyperledger-based microservice architecture with possible extensions**

Holochain, on the other hand, could be an interesting extension towards distribution of the blockchain to the end-users, i.e. making the end-user terminals (phones) nodes in the blockchain network. This could be interesting for censorship-resistance scenarios, but also for situations where the economy is more informal. For example, in poor neighbourhoods or in contexts where there are migrants, or aid being delivered to migrants. This aspect is farther in the future not only because of the difficulty the realisation of such a financial model would entail but also because the Holochain framework is farther behind in development relative to many others.

2.3 INTERLACE Open Shared Architecture

The final view of the architecture is based on the Corda-Network example,[2] which we find deeply interesting and innovative. Its most important aspect, which we plan to emulate, is to separate the Figure 2.2 view into the three layers shown in Figure 2.3 (the orchestration layer is more technically than conceptually relevant so we do not focus on it). This is done not only from the unsurprising functional stack point of view but, more importantly, in terms of the controlling legal entities.

More specifically, the bottom Infrastructure Layer becomes a semi-public, shared, and *neutral* infrastructure as a permissioned blockchain that is a direct extension of the INTERLACE blockchain. Its legal personality could be a foundation, as Corda have done, or similar non-profit entity. However, it will need to be economically self-sustaining; therefore, it will charge for Identity and Notarisation services. The result of this separation is that the bottom, persistence blockchain layer is relatively

[2] https://corda.network/

lightweight – and therefore of relatively modest operational management requirements. Most the complexity will be housed in the middle layer.

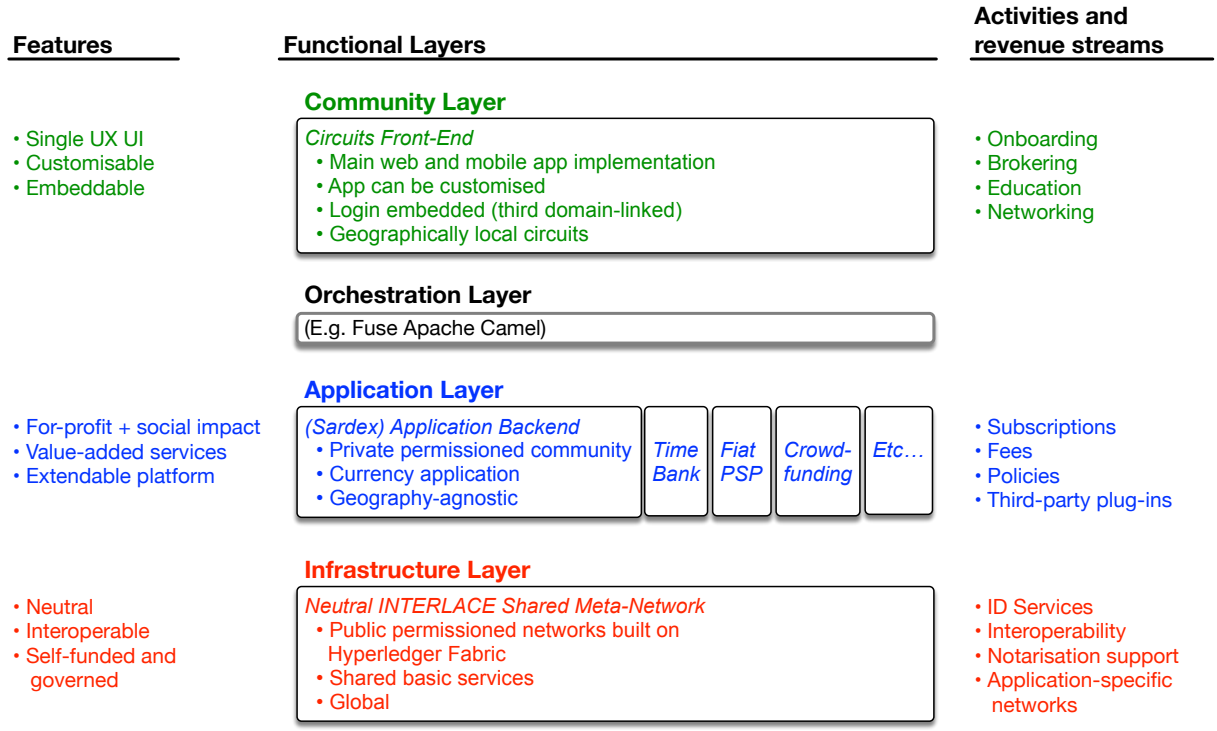

Fig. 2.3: **Long-term view of the Sardex open shared architecture stack**

The Infrastructure Layer will remain open source. Access will only be given to legal entities, not to natural persons. Finally, this layer will be accessible also to regulators and tax authorities, and will comply with the relevant directives. From the point of view of governance, the Infrastructure Layer will be composed by different kinds of nodes, the first of which will be an Orderer, Identity Certificate Authority, and Notarlsation. But as new nodes join they will take these functions, on rotation, or other functions like Endorser Peer and so forth. These nodes could be run by participating company according to an agreed-upon policy, or by the founding members of the foundation, for example the original members of the INTERLACE project.

The middle, Application Layer is where the proprietary business logic of applications like Sardex will be housed, i.e. all the microservices shown in Figure 2.2. In the case of the Identity service, it will be split up in order to put the most basic certification aspect in the Infrastructure Layer while the GDPR-relevant user profile data plus other application-specific information will be held in the Application Layer. This layer will also host other applications, shown in the figure as Crowdfunding, normal fiat money Payment Service Providers, and so forth. The implication is that the neutral infrastructure layer will not be limited to credits: because it will not perform deep packet inspection it will mediate whatever the overlying applications will want to transfer. At this layer the architecture can be regarded as centralised since Sardex is the single owner running its proprietary application, even if it might become global.

The third layer is the community layer. This is where localism and the social and cultural dimensions will be represented and protected. This is where both legal entities (mostly but not exclusively SMEs) and natural persons/citizens/consumers will be able to form communities and interact commercially with each other, following the current Sardex model and approach. Each circuit will be run by a 'Partner', i.e. a for-profit or non-profit company that wishes to set up a Sardex-like circuit in their region.

The next aspect of the architecture to be considered is the interfaces. The interface between the Community and the Application layers will be controlled by the latter through a number of possible GUIs and an appropriate API to support different kinds of Admin access. The managers of the circuits will have a GUI into some of the services of the application, whereas SMEs and natural persons will need to go through their community's GUI/skin. Access will be clocked and charged according to the principles of mutual credit we are currently developing for scalability and that are briefly outlined in deliverable D4.2 [6]. For example, the middle-layer application may be partly owned by the circuits which, in turn, may be partly owned by their own users.

The interface between the Application and Infrastructure layers is where the 'permissioned' nature of the blockchain is most evident. Depending on the type of company requesting access, different kinds and level of access can be granted. For example, regulators will have a different kind of access from an SME.

Chapter 3

Conclusion

Paolo Dini

The question of regulating access is linked to a fundamental aspect of this architecture that has not been mentioned yet. Specifically, it opens the possibility to embed the principles of mutual credit in the infrastructure and the protocols themselves. We make this point in the Conclusion because it highlights the culmination of a journey of increasing self-awareness, self-determination, and appropriation of the technological means that support the financial infrastructure of the Sardex circuit.

The Sardex founders were aware from the beginning that their objective was a radical financial innovation in the service of the local real economy. This can therefore be regarded as a "political" project, not in the sense of party politics but in the wider sense of 'The Political' as defined for example in the Heteropolitics project:[3] *The Political is the deliberate, i.e. conscious, choice to form or influence social relations.* The Sardex founders were interested in improving social and economic relations in their territory, but were keenly aware that an explicitly party-political movement would not have been appealing to the great majority of Sardinian SMEs. Thus, they decided to act at the level of a different financial model.

Insofar as a financial model enables and influences economic behaviour and interactions and is encoded in a technology platform, it can be regarded as a form of infrastructure. Because the initial and still-functioning Sardex transactional platform (which INTERLACE aims to replace) is based on a fairly traditional relational database, the original Sardex innovation was ultimately an innovation in *financial architecture* rather than technological architecture. The system architecture that we have presented and discussed in a series of deliverables (D2.1 [1], D2.2 [2], D3.1 [4], D3.2 [5], D4.2 [6]) is nothing more than the consequence of this overarching requirement, coupled with the requirement that the system be scalable and sustainable in the long term.

To contextualise the architectural choices around access and mutual credit-specific protocols, it is helpful to view the actions of the Sardex founders and the overarching objectives of the INTERLACE project through the lens of Andrew Feenberg's Critical Theory of Technology (CTT) [3]. Feenberg starts with the assumption that, whether we like it or not, technology embodies our cultural values. As shown in Figure 3.1, Feenberg maps the philosophy of technology field with two axes: in one direction, technology is regarded as either value-neutral or value-laden; in the other direction, it is regarded as either human-controlled or autonomous. The top-left quadrant views technology as autonomous and value-neutral, like Marx's Determinism. The top-right quadrant is populated by people who are optimistic about technology and believe it is neutral. The bottom-left quadrant is populated by Luddites, who fear technology because they regard it as both laden with destructive values and autonomous. INTERLACE belongs firmly in the lower-right quadrant, where our awareness that technology is value-laden together with the belief in our ability to control it leads to a responsible engineering design ethos that can in some sense "domesticate" technology by making the technologists accountable to the users of their creations. The importance of bodies like the Internet Governance Forum[4] is ultimately founded on this assumption of responsibility of technology towards society. The foundation or similar body that will eventually control the Infrastructure Layer of the INTERLACE open shared architecture will be organised along similar principles, but with a narrower focus on enabling SMEs through local circuits.

[3] https://heteropolitics.net
[4] https://www.intgovforum.org/multilingual/content/igf-2018-0

Technology is:	Autonomous	Humanly controlled
Neutral	Determinism *(Marx)*	Instrumentalism *(optimistic)*
Value-laden	Substantivism *(pessimistic)*	Critical Theory *(Feenberg)*

Fig. 3.1: **The philosophy of technology field according to Feenberg [3]**

From this point of view, therefore, the long-term view of the Sardex system is that it will rely on an infrastructure that will enable and support certain kinds of business behaviour while discouraging or making impossible other kinds. Thus, from the CTT viewpoint the Infrastructure Layer is *not* neutral at all, and consciously so. In the previous chapter this layer is characterised as neutral from the point of view of private ownership and competition: it is neutral and open to any legal entities that wish to use it, but the services it will provide are constrained by the non-neutral values that it will embed, namely to support and encourage the real economy and to discourage or block out the financial economy (as it is constituted today).

The clearest examples of how this can be achieved at the level of requirements is to encode the zero-interest and non-convertibility principles into the blockchain protocols/smart contracts that act in this bottom layer. It is not yet clear whether specific access restrictions (to e.g. securities traders) should also be enforced, but it may be sufficient (and would be more elegant) simply to disable certain types of financial operations. For example, any mutual credit application that wants to use the blockchain Infrastructure Layer will need to comply with both requirements, but a fiat PSP may need to be able to convert between different currencies. However, even a PSP will not be able to use the blockchain to *store* fiat money or other crypto-currencies, because this blockchain will not be cryptocurrency-based and will not (cannot) be a bank. It can store credits precisely because credits are not regarded, by the banking regulators, to be on the same footing as fiat and, therefore, they are PSD II-exempt.[5]

Similarly, multiple mutual credit system will be able to use the bottom layer, not just Sardex S.p.A., but convertibility between different credit units will not be allowed; this is to prevent the migration of currency exchange speculative practices to the exchange between different credit units. However, if anyone mutual credit application happens to straddle different currency areas, as Sardex is planning to do, then of course an exchange mechanism will be needed given that each credit unit will be pegged 1-1 to its local fiat currency. Another example of a political principle embedded in the protocol, in this case, is the fact that there will be only one exchange rate in both directions between any two currency areas. Finally, these currency exchange rates are an example of the kind of information that will be publicly accessible to anyone, not just the legal entities that have registered to use the permissioned blockchain.

In the long term, we hope that this architecture will enable the scaling of the Sardex mutual credit system to support SME trade worldwide in a way that is transparent and accountable, even if private as concerns business-confidential information and GDPR-regulated personal information; that will be compliant with all banking and PSP regulations; and that will encourage an ecosystem of similarly-minded legal entities in a constructive balance between cooperation and competition in the real economy.

[5] https://ec.europa.eu/info/law/payment-services-psd-2-directive-eu-2015-2366_en

References

References

1. P Dini, E Börger, E Hirsch, T Heistracher, M Cireddu, L Carboni, and G Littera. *D2.1: Requirements and Architecture Definition*. INTERLACE Deliverable, European Commission, 2017. URL: https://www.interlaceproject.eu/.
2. P Dini, G Littera, L Carboni, and E Hirsch. *D2.2: Iterative Architecture Refinement*. INTERLACE Deliverable, European Commission, 2018. URL: https://www.interlaceproject.eu/.
3. A Feenberg. *Questioning Technology*. London and New York: Routledge, 1999.
4. E Hirsch, T Heistracher, P Dini, E Börger, L Carboni, M L Mulas, and G Littera. *D3.1: First Demonstrator Implementation*. INTERLACE Deliverable, European Commission, 2018. URL: https://www.interlaceproject.eu/.
5. E Hirsch, T Heistracher, P Dini, E Börger, L Carboni, M L Mulas, and G Littera. *D3.2: Final Demonstrator Implementation*. INTERLACE Deliverable, European Commission, 2018. URL: https://www.interlaceproject.eu/.
6. G Littera, G Dini, and P Dini. *D4.2: Report on Field Tests and Commercialization Plan*. INTERLACE Deliverable, European Commission, 2019. URL: https://www.interlaceproject.eu/.

Appendix: Additional Functional Requirements and Business Logic Model (2018)

Eduard Hirsch and Paolo Dini

In order to see how to manage a more challenging use case based on reading historic data placed on the chain, we model the "delta-debt" function as an additional requirement. This example could help foresee possible challenges for more complex tasks. The implementation details for the other requirements covered can be found in D3.2 [5].

A.1 Requirement: Debt Record Tracking

Sardex sets the credit line of the circuit's SME users based on their turnover as well as on their track record. Thus, the balance of an account, as described in the requirements specification deliverables D2.1 and D3.1 [1, 4], can go negative (with a 0% interest rate) up to a maximum amount that varies for each member company. It is a Sardex contractual requirement that such a negative amount or debt be "paid back" within 12 months of when it was incurred. The debt is not towards Sardex S.p.A. (Sardex S.p.A. is not a bank) and it is not bilateral towards a single other member. Rather, the debt is towards the circuit as a whole. Therefore, the indebted company can pay back or recover its debt simply by selling its products and services to any other members, at least for the amount of the debt and within 12 months of incurring it. In general, each and every transaction that increases the debt of an already negative balance needs to be recovered within a separate 12-month window.

More precisely, any transaction that increases the debt of an account triggers a recording of that newly created portion of debt and causes the system to allocate a separately handled due date 12 months later for its repayment. On the other hand, if an account receives a positive amount of credits (due to the sale of a product or service), then the "debt-portions" which that account may have accrued up to then are paid back in sequence, starting from the oldest unpaid one.

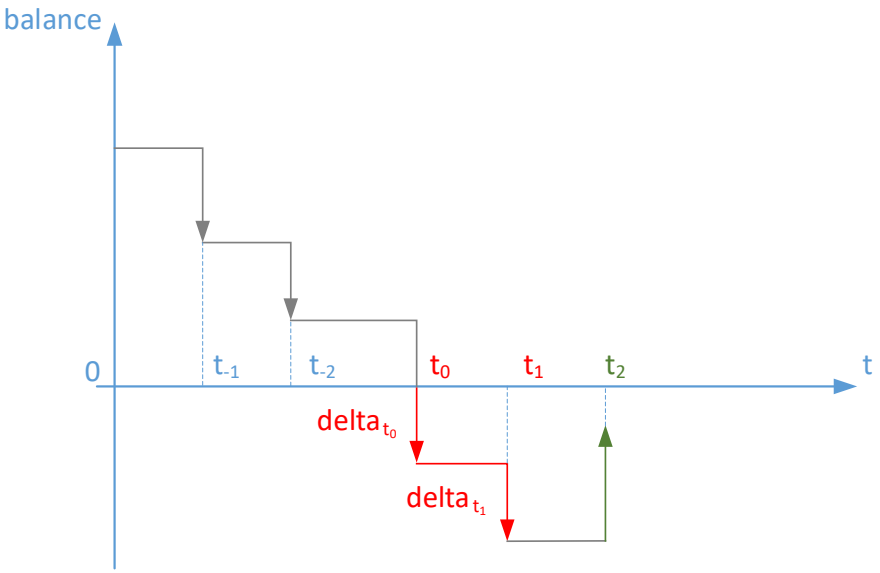

Fig. A.1: **The delta-debt progression**

Figure A.1 illustrates a transaction flow for a single account with 5 transactions and their timestamps named t_i, where i covers the interval $[-2, 2]$ in discrete steps of 1. Two timestamps t_j and t_k with $j > k$ imply that t_j is older than t_k. Let's assume a transaction at t_0, as illustrated in the figure, which turns a positive balance into a negative value. That event then defines a starting timestamp at t_0.

This transaction at t_0 creates a debt whose value is recorded in the variable *balance* after the transaction. In other words, the value of *balance* before the transaction is reduced by the transaction *amount*, and it becomes a debt because, before the transaction, $balance \geq 0$ and $amount > balance$. The debts $debt_{t_0}$ and $debt_{t_1}$ created by this transaction and by a possible further transaction, respectively, are represented in Figure A.1. Furthermore, these debts are treated as positive values. Thus, we define a debt created at t_i as

$$debt_{t_i} = \begin{cases} |balance_{t_i} - amount_{t_i}| & \textbf{if } i = 0 \textbf{ and } balance_{t_0} \geq 0 \textbf{ and } amount > balance \\ amount_{t_i} & \textbf{if } i > 0 \textbf{ and } balance_{t_i} < 0 \\ 0 & \textbf{else} \end{cases}$$

By declaring $txid_{t_i}$ as the id for a transaction which is executed at time t_i and with *accid* as the account id, we are defining a so-called *deltaDebt* with index i as the following tuple:

$$deltaDebt_i = (t_i, txid_{t_i}, debtPos_{t_i}, accid), \tag{1}$$

The $amount_{t_i}$ is not needed any more once the new $debtPos_{t_i}$ has been computed, namely as $debtPos_{t_i}$ reduced by any amount **received** by the same account.

The creation of a *deltaDebt* can be described as follows. Note that it will be called only when some debit is created or an existing credit increased.

```
CreateDeltaDebt(txid, currentDate, from, amount) =
  let balance = balanceOf(from)
    if (balance − amount) < 0 then
      let debt =
        { |balance − amount|   if balance > 0
        { amount               else
      let deltaDebt = (currentDate, txid, debt, from)
        WriteDeltaDebt(deltaDebt)
```

In order to pay back an open debt, at every transfer the *deltaDebt*s (if there are any) need to be checked for possible clearances:

```
ClearDeltaDebt(txid, to, amount) =
  if amount ≠ 0 then  // no debt change when amount = 0
    let openDeltas = SelectOpenDeltaDebtsFor(to)
      ClearDebt(openDeltas, amount)
where
  ClearDebt(openDeltas, amount) =
    if openDeltas ≠ ∅ then
      let oldestOpenDebt = SelectOldestDebt(openDeltas) // NB. oldestOpenDebt ≠ undef
      if debtPos(oldestOpenDebt) ≥ amount then
        debtPos(oldestOpenDebt) := debtPos(oldestOpenDebt) − amount
      else
        let restAmount = amount − debtPos(oldestOpenDebt)
          debtPos(oldestOpenDebt) := 0
          ClearDebt({d ∈ openDeltas | d ≠ oldestOpenDebt}, restAmount)
```

In this ASM example we assume that SelectOpenDeltaDebtsFor is returning a set of open *deltaDebt* entries from the persistence layer where *debtPos* of each *deltaDebt* is bigger than 0. SelectOldestDebt selects the *deltaDebt* of a corresponding set with the minimum timestamp. Further, SelectOldestDebt is only defined if *openDeltas* is not an empty set.

debtPos(*oldestOpenDebt*) reads or writes the pending amount for a particular *deltaDebt* (here: *oldestOpenDebt*).