WP2

Iterative Architecture Requirements and Definition

Deliverable D2.1

Requirements and Architecture Definition
INTERLACE Project (Grant no. 754494)

Contract Number: 754494
Project Acronym: INTERLACE

Deliverable No:  D2.1
Due Date: 31/07/2017
Delivery Date: 20/10/2017 (undated post-submission version, with additional corrections in red font)

Author: Paolo Dini (UH), Egon Börger (UNI PASSAU), Eduard Hirsch, Thomas Heistracher (SUAS),
Massimo Cireddu, Luca Carboni, Giuseppe Littera (SARDEX)

Partners contributed:
Made available to: Public

Versioning

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Name, organization</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>15/06/2017</td>
<td>Paolo Dini (UH)</td>
</tr>
<tr>
<td>2</td>
<td>15/07/2017</td>
<td>Egon Börger (UNI PASSAU), Massimo Cireddu, Luca Carboni, Giuseppe Littera (SARDEX)</td>
</tr>
<tr>
<td>3</td>
<td>21/08/2017</td>
<td>Egon Börger (UNI PASSAU), Eduard Hirsch (SUAS), Massimo Cireddu, Luca Carboni, Giuseppe Littera (SARDEX)</td>
</tr>
<tr>
<td>4</td>
<td>31/08/2017</td>
<td>PAOLO DINI (UH), EDUARD HIRSCH (SUAS), LUCA CARBONI, MASSIMO CIREDDU, GIUSEPPE LITTERA (SARDEX)</td>
</tr>
<tr>
<td>5</td>
<td>08/10/2017</td>
<td>PAOLO DINI – INTEGRATED FINAL COMMENTS FROM INTERNAL REVIEW</td>
</tr>
</tbody>
</table>

Internal Reviewer: Chrystopher Nehaniv (UH)

This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License.
Abstract

This report describes the high-level requirements of the new decentralized transactional platform for the Sardex mutual credit system. The effort has been divided into two parts: first, the high-level functional requirements (business logic) are specified, modelled, verified, and implemented in a production programming language; second, the same process is applied to the backend infrastructure, which will be based on the most suitable blockchain technologies and architecture based on the economic, financial, and governance framework that SARDEX is in the process of defining. This report provides an initial overview of the architecture requirements according to the arc42 method, and then provides a formalization of the specification and modelling of the business logic using the Abstract State Machines method and the CoreASIM framework for executable Abstract State Interaction Machine models.
# Table of Contents

1 Introduction

2 High-Level Architectural Requirements and Documentation
   2.1 About INTERLACE
   2.2 Introduction
      2.2.1 Goals
      2.2.2 Context and Previous Work
      2.2.3 Requirements Overview
      2.2.4 Quality Goals
      2.2.5 Stakeholders and their roles
   2.3 Solution Strategy
      2.3.1 Strategy Steps
   2.4 Risks and Technical Debts
   2.5 Glossary

3 Functional Requirements and Business Logic
   3.1 Core Payment Operations
      3.1.1 Signature elements of B2B Operations
      3.1.2 Behaviour for Credit Operations
      3.1.3 Behaviour for Debit Operations
   3.2 Account History and Balance Operations
   3.3 User Operations

References
Chapter 1

Introduction

Paolo Dini and Giuseppe Littera

The INTERLACE project is developing a blockchain-based transactional platform for use by the Sardex mutual credit system. Sardex S.p.A. (SARDEX) has been operating successfully an electronic, B2B, zero-interest mutual credit system on the island of Sardinia since 2009. The Sardex system (also known as Circuito di Credito Commerciale) challenges prevailing notions about the nature of money, and the financial and economic autonomy that a relatively poor region can aspire to, because it enables local economic actors (SMEs in particular) to trade with each other in a trustful and circular fashion with a unique digital trade credit unit. It does this by monetizing the spare capacity of the local economy in the form of mutual, and taxable, credit between participating companies, at zero interest, on a strong basis of trust, solidarity, and local cultural identity [4, 3, 11]. The deeply innovative nature of this system is to distribute to the circuit members the power to create credit money (sardex credits, where 1 credit = 1 Euro), and thereby provide an alternative to credit money creation through bank loans. However, the Sardex transactional platform is currently centralized, which challenges the long-term scalability, sustainability, and governance of the system because the governance and management of the circuit are all held by the central credit-clearing entity (SARDEX).

The specification and implementation of the new architecture are based on the Abstract State Interaction Machines (ASIMs), developed and implemented in the platform CoreASIM by the BIOMICS project [8, 7, 10] as extensions of Börger and Stärk’s [1] Abstract State Machines (ASMs) and of the CoreASM environment. The ASIMs are based on realizing the BIOMICS mathematical framework for Interaction Machines (IMs) that dynamically and recursively grow and change their components and network topologies to deploy/reabsorb resources in response to interactions and computational needs [5]. Relative to the ASMs, ASIMs are fully asynchronous, concurrent, and communicating, so they can run on different servers and communicate over the network to validate transactions. The ASIM approach is fundamentally important for SARDEX as a company because it guarantees verifiability, validation, and efficient change management (to manage requirements creep as well as new emerging functionalities) through rigorous mathematical formalization of the specification at the level of requirements capture and a rigorous process of iterative refinement down to the level of the code of choice. Any of these levels of abstraction is executable by an interpreter (built into CoreASIM), so at each level of the refinement process the current implementation level can be verified against requirements.

The number of new cryptocurrencies is increasing very rapidly, along with the variations in the technologies that support them. This very volatile technology landscape is causing us to focus first on the new economic and governance model for sardex.net, which will provide the high-level requirements for the new blockchain architecture and will, therefore, enable us to narrow down the number of possible frameworks and technologies to draw from. In the meantime, we have begun the non-trivial task of migrating the current platform functionality towards the new model. The first step in this process has been to begin formalizing the circuit’s business logic as ASM models.

---

1 Cyclos: http://www.cyclos.org/products/
2 http://biomicsproject.eu/
4 As of 28/08/17 there were 865 currencies listed on https://coinmarketcap.com/, up from 851 a few days earlier.
This report is organized as follows. Chapter 2 provides a high-level view of the architecture and its documentation following the arc42 method.\textsuperscript{5} Chapter 3 provides a first collection of functional requirements of the system modelled as ASMs. These reflect the business logic of the \textit{current} system. This model will be instantiated in CoreASIM in order to execute it with a fictitious set of inputs and verify its operation, after which it will be implemented in a language of choice (probably Java).

A second specification and modelling effort will follow, to reflect in the blockchain “backend” the new governance and financial/economic model once it has been completed. This second model and its implementation will be reported in the next architecture deliverable (D2.2) at Month 12.

\textsuperscript{5} http://arc42.org/
Chapter 2

High-Level Architectural Requirements and Documentation

Paolo Dini, Eduard Hirsch, Giuseppe Littera, Luca Carboni, Massimo Cireddu, Thomas Heistracher

2.1 About INTERLACE

The objective of INTERLACE is to use the Abstract State Interaction Machines framework (CoreASIM)\(^6\) open source output of the FP7 FET project BIOMICS\(^7\) to develop a decentralized transactional and ledger architecture demonstrator for B2B mutual credit.

2.2 Introduction

2.2.1 Goals

Currently Sardex uses a payment platform which offers solid financial transaction facilities in support of B2B trade. However, the architecture has been built using a monolithic centralized approach which is not scalable as the system grows in size. In this context, the ‘system’ refers both to the 3500 users in Sardinia as well as the approximately 3000 members across the other 11 circuits in the other Italian regions and the 1500 individual users (members of the Business to Employee, B2E, programme). The aim of the platform redesign is to move to a decentralized (within INTERLACE) and ultimately to a completely distributed architecture which is able to scale far beyond the capacity of the current implementation. Figure 2.1 shows the current and the first step in decentralization.

It is too early to imagine what form the fully distributed architecture will take, especially since blockchain technology is being innovated and is diversifying as different forks at incredible speed.

\(^6\) http://biomicsproject.eu/news/135-icef
\(^7\) http://biomicsproject.eu/
Equally important is the principle according to which the technological architecture should be seen as reflecting and responding to the requirements set by the economic, financial, and governance model, and not the other way around. Since the economic, financial, and governance model is also evolving, it is more important to work towards a tight coupling (e.g. a compiler) between a formal but agile specification framework and the corresponding implementation than to have all the architectural answers now.

As stated in the Introduction, we are using the ASM method [1] to develop executable models of the functional requirements. These models will be implemented in the CoreASIM framework,8 where they can be validated before implementing them in a production environment (e.g. in Java).

2.2.2 Context and Previous Work

The INTERLACE proposal describes a blockchain architecture based on the Open Transaction protocol (OTX) [6] as an intermediate solution between fully centralized and distributed architectures. OTX involves a pool of Auditor nodes to validate the transactions executed by each Notary node. The initial conception of the INTERLACE architecture was to use only one central Notary, as a first step from the current centralized server towards a more distributed architecture. For the persistence layer we originally examined the Lightweight Cryptocurrency Ledger (LCL) [13] as an alternative to the Bitcoin blockchain. The idea of LCL is to store minimal information on the current state of the whole ledger in each block rather than requiring nodes to carry the whole history of block deltas as the Bitcoin blockchain does. This approach achieves continuity with the existing solution while also enabling scalability to multiple circuits (multiple Notaries) under the same mathematical and computational framework.

Rather than replicating the functionality of Cyclos 4 in a decentralized or distributed manner by implementing the OTX-LCL concepts from scratch, it makes sense to take advantage of the staggering levels of investment currently being made in blockchain technologies by several banking and industry consortia, especially since the best ones are open source, and build on existing frameworks. As of April 2017, just before INTERLACE started, the two most likely candidates for our purposes were IBM’s Hyperledger Fabric9[2] and Corda.10 Together, they can be described as bringing together the principles of OTX and LCL with the smart contracts of Ethereum.11 However, an important difference is that these blockchain implementations are permissioned, they are not open (permissionless) like the Bitcoin or Ethereum blockchains. Although this suits the early decentralised implementation of the INTERLACE blockchain, it opens up governance questions that are currently being examined by SARDEX. First among these question is the possibility of establishing two separate legal entities:

- Sardex S.p.A. (SARDEX) will maintain its current ownership structure.
- A new non-profit legal entity, which we can refer to for now as Circuit Coop, or more loosely as sardex.net, may be formed in the near future to begin devolving the ownership of the commons12 to the circuit members.

From a purely functional point of view, Hyperledger enables a given node to belong to multiple blockchain networks. This is of interest to us because in the governance framework SARDEX is defining at least two units to be stored on the blockchain are envisaged: SRD credits and a new meritocratic reward points system called ‘Proximity’, implemented with tokens that are earned through altruistic behaviour and dubbed ‘π’. Corda is interesting because it separates what Hyperledger calls ‘chaincode’,

---

10 [https://www.corda.net/](https://www.corda.net/)
11 [https://www.ethereum.org/](https://www.ethereum.org/)
12 The Sardex circuit commons have not been defined yet, but they will constitute a new basis for shared ownership by all the members. An example could be solar panels owned by the coop and providing renewable energy that can only be bought with credits.
which implements the smart contracts, from the persistence layer. Thus, Corda comes with a ‘business
flow’ layer that provides greater flexibility to adapt to the great diversity of actors in the possible
future scenarios, e.g. partially overlapping networks involving trade, communications, and renewable
energy, and across different regions or even countries.

The functional requirements addressed in this report cover the core functionalities like credit and
debit operations but exclude and intentionally hide implementation details about how transactions
are actually processed on the backend servers. The reason is to divide the development work into
simpler stages rather than achieving the redesign of the full stack in one step. This is possible thanks
to CoreASIM acting as an interpreter of executable ASIM models, which can invoke interface object
to a simplified user interface and to the backend. In this manner the high-level specification and
requirements can be verified before the implementations of either the new front-end and backend are
performed.

2.2.3 Requirements Overview

The requirements have drifted over time due to two causes. First, the technology is changing very
quickly, so what we envisaged in terms of technologies at the time of the proposal writing is now
obsolete; second, the governance, economic, and financial model of Sardex is also evolving, so the
high-level requirements themselves have changed relative to a year ago. Therefore, the requirements
below will be instantiated in a specific development strategy to be detailed later in this chapter.

<table>
<thead>
<tr>
<th>Req</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>Needs a transaction layer and a persistence layer</td>
</tr>
</tbody>
</table>
| R2  | Both layers must be extensible and scalable to a (global) distributed architecture,
but must start decentralized in their initial (local) implementations |
| R3  | Should be faster than Bitcoin, so a lightweight ledger (fragmented blockchain) approach is preferred |
| R4  | Must be able to support intra-trade and inter-trade between multiple circuits. For example, the
different circuits could be:
  • different Mutual Credit (MC) circuits (Sardex & Tibex)
  • different types of networks (MC circuit and Renewable Energy (RE) network) |
| R5  | If possible, reuse existing open source solutions and frameworks, within our own customized
ASM/ASIM framework |
| R6  | Chain code (smart contracts code) should be separate from transaction layer for greater speed
and efficiency |
| R7  | Inter-circuit operations must avoid falling under the European PSD2 Directive |
| R8  | The Sardex blockchain should not have a native token: this is in order to separate unit of account
and medium of exchange from the operation of platform (i.e. no mining, as in Bitcoin) and avoid
seignorage (as in Ripple/XRP) |
| R9  | Smart contracts code can be Turing-complete. The undecidability of Turing-complete languages does
not prevent the ability to prove specific properties of specific programs. In the ASM methodology,
provability is refined along with the specifications as part of the iterative refinement process,
down to the actual code. |
| R10 | Support the needs of Industry, Academia, NGOs, non-profits, Social Movements |
| R11 | Platform must involve the current regional MC systems and a reward and digital asset system called
Proximity and whose unit of account is called \( \pi \) |
| R12 | Proximity involves a reward points system whereby \( \pi \) s are awarded to users on the basis of
behaviour that is beneficial for (their local) MC system. Gaining \( \pi \) translates into the ability to
trade farther away from the user’s geographical location. Upon reaching a certain threshold,
the user is allowed to trade inter-circuit. |
| R13 | The new platform architecture should be closed, i.e. ‘permissioned’, both for the regional networks
and for Proximity: Open (permissionless) DLT networks like Ripple/XRP could not prevent
external actors from speculating on Sardex digital assets like \( \pi \). Private (permissioned) DLTs
like Chain Core/Ivy (compatible with PSD2) are preferred |

Table 1: Summary of initial high-level requirements
2.2.4 Quality Goals

The new system will need to separate different functions that are needed in a circuit, such as transactions, search, ads, entities, and so forth, because they all have different service-level agreements (SLAs). Therefore, a microservice architecture is the best approach.

In the current architecture definition process, it is also important to avoid dependencies on specific technologies. This is because we are in the middle of a rather chaotic period of innovation in payment technologies. This is over and above an even more intense period of innovation in the blockchain space, so we are talking about two layers of very fast-moving technological innovations that we need to keep abreast of. Thus, we must be clever in making the right choices and in leaving open the possibility for changes in the technology we ourselves develop.

Open source is a great philosophy and it is responsible for an amazingly fast evolution in Information Technology, but there is a cost. The success of an open source project depends in large part on the ability of the project to enthuse potential contributors. Therefore, we must not forget that our work must be attractive to other passionate developers. Also, great open source software has great documentation. In summary, when we write code, we must act responsibly in using best practices, standards, and white papers. This help whoever wants to contribute at the start, and will ultimately help everyone develop high-quality software.

The stakeholders listed in Section 2.2.5 need to be sure that the Sardex S.p.A. ledger application is highly available and reliable in order to act as a proper mutual credit system. Here you will find a list of quality goals for the architecture whose fulfilment is of highest importance to the stakeholders:

1. Highly decentralized architecture which can be developed to a fully distributed architecture.
2. Allow reliable transaction which are immutable and traceable.
3. High availability and operational as near real-time system.
4. Enforce high security standards.

2.2.5 Stakeholders and their roles

Common abbreviations for circuit stakeholders are:

- B: Business
- C: Consumer
- E: Employee

<table>
<thead>
<tr>
<th>Role/Name</th>
<th>Contact</th>
<th>Expectations</th>
</tr>
</thead>
<tbody>
<tr>
<td>B2B</td>
<td>Participating Companies</td>
<td>Business-to-business transactions and interaction. Find other Sardex members, advertise its business inside the network, pay and receive credits from others members, manage its accounts.</td>
</tr>
<tr>
<td>B2E</td>
<td>Employees of Sardex member companies</td>
<td>Receive a part of his salary in Sardex credits (for example bonus, benefits, etc.). Find Sardex companies where she can spend her credits. Manage her account.</td>
</tr>
<tr>
<td>B2C</td>
<td>Generic Consumer</td>
<td>Payments to Participating Companies. Find Sardex companies participating in the B2C programme, receive credits as cash-back for payments in Euros, spend her credits, and manage her account.</td>
</tr>
<tr>
<td>Sardex-Admin</td>
<td>Sardex S.p.A. Employee</td>
<td>Configure and maintain the infrastructure. Support members and different kinds of Sardex admins (i.e. Brokers, Community Trade Advisors) with technical issues. Help members collaborate and close deals.</td>
</tr>
</tbody>
</table>


<table>
<thead>
<tr>
<th>Role/Name</th>
<th>Contact</th>
<th>Expectations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sardex-Manager</td>
<td>Sardex S.p.A. Employee</td>
<td>Run evaluations and manage the cooperation platform. Monitor daily/monthly/yearly metrics of transactions, members and monetary parameters.</td>
</tr>
</tbody>
</table>

2.3 Solution Strategy

The strategy of how to get from a monolithic working implementation to a decentralized and later a fully distributed ledger application is broken down in several steps, as outlined below.

Since the original ASMs work only in their own scope, it is not possible to create a decentralized or fully distributed environment with them. The solution for the INTERLACE project has been to use a special extension of the ASMs named Abstract Interaction Machines (ASIMs) developed by the BIOMICS project, and a corresponding runtime environment called Interaction Computing Execution Framework (ICEF) \(^\text{13}\). As explained on the ICEF webpage:

This framework extends the original CoreASM modelling and execution framework to enable the specification and execution of distributed and concurrent ASMs. The ICEF was developed in the STREP project BIOMICS which was financed by the European Commission in FP7 from October 1st, 2012 until March 31st, 2016. ICEF enables asynchronous execution of ASMs. It uses and enhances the CoreASM execution engine to support communicating and interacting ASMs: CoreASIMs. Further, ICEF replaces ASM with BSL which offers additional language primitives specifically designed to define the behaviour of biochemical systems. This code introduces a restful API to control the BIOMICS wrapper (brapper) which can host several CoreASIM instances and enables networked CoreASIM. It also introduces a manager which orchestrates several ASIMs to allow the execution of interaction computing simulations.

Although the additional language primitives offered by ICEF might not be needed, the CoreASIM implementation will be crucial for realizing the INTERLACE strategy.

2.3.1 Strategy Steps

The development strategy between now (M4) and the end of the project at M18 is as follows:

1. Define the functional requirements of the business logic using the ASMs formal description language (this report).
2. Translate the formal description to a working demo environment using ICEF/CoreASIM.
3. Connect the business logic modelled with ASMs and implemented in CoreASIM to the real world. That means use the interaction capabilities of the framework to connect it to the existing legacy application used by SARDEX, Cyclos 4. This existing working architecture is summarized at a high level in Figure 2.2.
4. Test if the application and the translation to CoreASIM work.
5. Translate the CoreASIM implementation to a real-world application by creating e.g. a JAVA application.
6. Develop a verification strategy and validate the implemented real-world application through component testing.
7. Select appropriate blockchain technologies for the next-generation infrastructure.
8. Develop ASIM specifications that model the distributed environment.
9. Adapt ASIM interfaces.
10. Adapt the real-world version of the application to the new infrastructure.
11. Implement the new back-end logic using Open Source frameworks.
12. Test/Validate with real users the new application against the new back-end.

Several decentralized and distributed models with dynamic growing and adaptive hierarchical network topologies from the BIOMICS project can be found in \([5, 10]\) and proof-of-concept implementations using ASIMs are described in \([10, 12]\). 

\(^{13}\) [https://github.com/biomics/icef](https://github.com/biomics/icef)
AWS-Hosted 3-Tier Cyclos 4 System Architecture

Legend:
AB = Amazon Bucket Object Backup Storage
ACL = Access Control List
AGS = Amazon Glacier Storage
AMI = Amazon Machine Image
AWS = Amazon Web Services
EIP = Elastic IP Address
RDS = Amazon Relational Database Storage
R = Amazon Replica DB Backup
VPC = Virtual Private Cloud

Fig. 2.2: Existing 3-tier Cyclos 4 architecture used by SARDEXX, based on Amazon Web Services, and showing some of the levels of redundancy
2.4 Risks and Technical Debts

The risks in the present development effort are mainly associated with the selection and adoption of a suitable blockchain framework. The number of open source frameworks to choose from is large and continues to grow. As the economic, financial, and governance framework converges to a stable model that is approved and accepted by the main stakeholders of the Sardex circuit, we will need to choose a suitable permissioned blockchain\textsuperscript{14} for the infrastructure, and at this point there is some uncertainty as to which is the best choice.

Another risk comes from the fact that the INTERLACE project has a small budget and cannot fund the whole development effort. Thus, a significant part of the work will need to be funded by SARDEX directly and/or by its partners. This issue will become more pressing towards the end of the first year of the project.

2.5 Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Centralized Architecture</td>
<td>Architecture that at a functional level has a central, single-owner server</td>
</tr>
<tr>
<td>Decentralized Centralized architecture</td>
<td>Centralized architecture where some of the control and/or is devolved to nodes other than the central one</td>
</tr>
<tr>
<td>Distributed Architecture</td>
<td>Architecture where the control and functionality are distributed equally to all the nodes</td>
</tr>
<tr>
<td>Blockchain Distributed ledger with different levels of replication depending on the variant</td>
<td></td>
</tr>
<tr>
<td>Permissioned Blockchain</td>
<td>Blockchain that requires login credentials. While at the infrastructural level the functionality is distributed onto multiple servers, the control is centralized</td>
</tr>
<tr>
<td>Permissionless Blockchain</td>
<td>Original distributed architecture of the blockchain (e.g. Bitcoin, Ethereum, etc.)</td>
</tr>
<tr>
<td>Abstract State Machines (ASMs)</td>
<td>Specification method that allows the definition of data structures that closely model the domain of interest, at the most appropriate level of abstraction, and with the required behaviour. Thus, the data structure concept includes also the definition of functions that operate on an ASM’s data structure, thereby changing its state. Since ASMs are rigorously defined and executable (interpretable) they can be used to define executable models that can be used to verify the requirements and validate models. Conceptually, ASMs can be thought of as analogous to finite-state machines, i.e. to a form of generalized automata. In the most general and abstract terms, ASMs can be thought of as a method to develop a customized programming language for a specific problem.</td>
</tr>
<tr>
<td>Abstract State Interaction Machines (ASIMs)</td>
<td>Extension of the ASM modelling framework to allow communication between different ASMs. ASIMs are ASMs that are equipped with a general scheduling mechanism and an interaction structure to distinguish between horizontal and hierarchical interaction, while supporting dynamically changing adaptive network topology. They can therefore be used to model an asynchronous, concurrent, distributed system with growing and changing structure.</td>
</tr>
<tr>
<td>CoreASIM</td>
<td>Extension of the CoreASM execution environment for ASMs.</td>
</tr>
<tr>
<td>Mutual Credit</td>
<td>A type of currency that structurally highlights the nature of money as a social relation of credit and debt. It is sometimes called ‘multilateral barter’, but this is not correct. While – like all currencies – mutual credit does enable multilateral exchange, it does not preclude profit margins, which are not easily quantified in barter. As credits perform the functions of medium of exchange, unit of account, store of value (across time), and means of payment (although they are not usually accepted for tax payments), they are a form of money.</td>
</tr>
<tr>
<td>SRD</td>
<td>Sardex credits. SRD do not accrue interest on either positive or negative balances and are not convertible with Euros.</td>
</tr>
</tbody>
</table>

\textsuperscript{14} Or similar, i.e. \url{https://wordix.inesctec.pt/wp-lightkone/wp-content/uploads/2017/07/lightkone-intro.pdf}
<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICEF</td>
<td>Interaction Computing Execution Framework</td>
</tr>
<tr>
<td>OTX</td>
<td>Open Transactions Protocol</td>
</tr>
<tr>
<td>LCL</td>
<td>Lightweight Cryptocurrency Ledger</td>
</tr>
<tr>
<td>SLA</td>
<td>Service-Level Agreement</td>
</tr>
</tbody>
</table>
Chapter 3

Functional Requirements and Business Logic

Egon Börger, Luca Carboni, Massimo Cireddu, Paolo Dini, Eduard Hirsch, Giuseppe Littera

We specify the core payment and related history operations of the INTERLACE network server, which as a starting point reproduce those of the current Sardex system. We do this at the functional requirements level of abstraction and in a component-based manner so that the resulting model can serve as abstract description of the current implementation but also as starting point for a new, blockchain-based implementation. Sect. 3.1 models two basic payment operations, Sect. 3.2 account history and balance operations, Sect. 3.3 user operations. Permission features and onboarding operations will be modelled in the near future and reported on in the next deliverable (D2.2).

3.1 Core Payment Operations

In this section we describe the interaction of the INTERLACE network server with users. We consider here only B2B operations, leaving the consideration of operations between a Company and either Employees or Consumers for a later phase. Sect. 3.1.1 explains the basic data types, Sect. 3.1.2 the credit and Sect. 3.1.3 the debit operation.

3.1.1 Signature elements of B2B Operations

The actors of B2B operations are companies (elements of the set Company) which interact with the INTERLACE network server on a request/response basis using various communication devices from whose technical details we abstract here. Therefore it becomes natural to describe the interaction of companies with the INTERLACE network server by SEND and RECEIVE actions of communicating Abstract State Machines (ASMs), the basic concept underlying Abstract State Interaction Machines (ASIMs)\(^{15}\), one for each participating company and one for the INTERLACE network server.\(^{16}\) We concentrate our attention in this section on modelling the actions the INTERLACE network server performs when triggered by requests sent to it by any company of the circuit (which are treated in Sec. 3.3).

We keep the communication mechanism abstract. SEND(msg, to: a) denotes the operation of sending the msg to agent a. Received(msg, from: s) is a predicate which is true when the message msg from the sender s is in the mailbox of the receiver. CONSUME(msg) denotes the operation of deleting the msg from the mailbox once it has been processed.

However, we should be aware that in modelling the Sardex transaction system communication takes place at two levels:

\(^{15}\) ASIMs are communicating ASMs which are equipped with a general scheduling mechanism and an interaction structure to distinguish between horizontal and hierarchical interaction as well as dynamic creation and reabsorption of components. These features have been defined to satisfy the requirements of Interaction Computing formulated in Deliverable 5.1 of the BIOMICS project [9]; these requirements have been shown to be satisfied by ASIMs (see Ch. 2 of Deliverable D5.2 [10]). For further details (in particular on the definition of the communication network structure, using channels and a routing component, and a resource manager by specialized communicating ASMs) and the implementation see \url{https://github.com/biomics/icef}.

\(^{16}\) Observe that in a distributed version of the Sardex system different instances of the system are run by different agents which all execute the same ASM program (or a program that has been obtained by adapting the basic program appropriately for a particular instance). Cyclos today is loosely coupled in the sense that one can have multiple applications running on multiple machines that share consistency with third-party utilities.
1. ‘from’ and ‘to’ as they apply to transfer operations between accounts
2. ‘from’ and ‘to’ as they apply to communications between ASIMs

Thus, for example, the following rule definition belongs to Level 1:

```
CreditPreviewReq((channel; from; to; amount; custFlds); mbr) = etc.
```

This is a rule that is invoked as part of an agent’s program. However, it could also be a part of a Level-2 communication. In the ICEF the more abstract notation `Send(msg, to: a)` mentioned above has been implemented as:

```
Send( ((channel; from; to; amount; custFlds); mbr),  to: TargetASIM,  subject = “CreditPreviewReq”)
```

This Level-2 message is channelled through the Mailbox of the SourceASIM and arrives into the Mailbox of the TargetASIM. The `from:/to:` notation does not describe a function and should be seen as something closer to a comment. The level it refers to should be clear from the context. It is used for the convenience of reading a rule where it only makes a parameter explicit that is used in the rule and anyway assumed to be part of the message in question.

We usually assume each `msg` ∈ `Message` to contain besides its `payload(msg)` also the information about its `sender(msg)` and `receiver(msg)`. Thus the parameter `from: c` in `Received(msg, from: c)` indicates that `sender(msg) = c`. Similarly, `to: c` in `Send(msg, to: c)` denotes that `receiver(msg) = c`.

The core payment operations are sent to the INTERLACE network server by companies `c` ∈ `SardexNet` which are members of the net.\(^{17}\) Each such company may have a number of `accounts`\(^{18}\) which we represent as elements of a set `Account(c)`. Each `account` has a well-defined `owner(acc) ∈ SardexNet` and is of some type `accountType(acc)` out of the set `AccountType` of possible account types:\(^{19}\)

```
AccountType = {credit, domu, fee}
Account = \bigcup_{c \in SardexNet} Account(c)
```

Therefore we name such accounts `creditAccount(c)`, `domuAccount(c)`, `feeAccount(c)` (names which are defined if `c` has the corresponding accounts).

There are two principal transfer operations, called Credit and Debit operation specified in Sec. 3.1.2 and Sec. 3.1.3.

At this stage of the specification we have not yet addressed security issues. For example, `sender(msg)`, `receiver(msg)`, or `msg` could be faked. Security will be addressed when we start specifying and modelling the distributed transactional platform. Any implications to the business logic models documented here will be examined at that time.

### 3.1.2 Behaviour for Credit Operations

A Credit operation is also called a push transfer. Its goal is to transfer an `amount` via a specific `channel` from one account to another. Sardex uses a `TransferType` concept which allows one to impose on the

\(^{17}\) We use for the datatypes evocative names which suggest their intended interpretation.

\(^{18}\) A company can have at most an account for each account type.

\(^{19}\) A credit account is a normal account that can have a positive or negative credits balance. A domu account is designed to allow the spending of credits for a mid/long-term investment — usually, but not always, to acquire buildings. Therefore, it can reach a relatively large negative balance, within the limit permitted by the credit line of the account. A fee account always has a positive credit balance, like a pre-paid card, and its balance is used to purchase one-off services, fixed-fee subscription services, or to pay for fees associated with fee-based transactions paid for from a different account (e.g. a credit account) held by the same user (although transactions between users belonging to the same circuit do not incur a fee, inter-circuit transactions do).
transfer operations certain conditions, including priorities. The parameters of a transfer type $tt$ which are relevant for the Sardex business logic are the following:\(^{20}\)

- the operation $\in \{\text{credit, debit}\}$,
- the channel $\in \{\text{phone, website, pos}\}$\(^{21}\) through which the interaction between the user and the INTERLACE network server takes place,
- the account type of the two involved accounts from, to $\in \text{Account}$,
- the groups of the two members involved: fromMbrGroup, toMbrGroup $\in \text{Group}$,
- conditions on the to-be-transferred amount,
- conditions on so-called custom fields.

$\text{CustField}$ is a set of typed variables, with or without parameters, in ASM terminology a set of 0-ary or $n$-ary functions (with $n > 0$) whose values are of an indicated type. They serve to encode customer information on the reason of a transfer, e.g. the number and date of the bill to be paid. For each transfer type $tt$ its $\text{custFields}(tt)$ (if there are any) are of two kinds, compulsory or optional. Optional fields do not affect the $\text{custFieldCond}$ of a transfer type. For a transfer to Match a transfer type $tt$ means in particular that its $\text{custFieldCond}(tt, \text{custFields}(\text{transfer}))$ evaluates to true, where $\text{custFields}(\text{transfer})$ denotes the values of the custom fields in the transfer. As a consequence if a transfer type $tt$ comes without or only with optional custom fields, then the $\text{custFieldCond}$ with $tt$ as first argument is empty, i.e. simply true.

We retrieve for each parameter the corresponding information from a transfer type $tt$ by an appropriate function:

\[
\begin{align*}
\text{name}: & \text{TransferType} \to \text{Identifier} \\
\text{oper}: & \text{TransferType} \to \text{Operation} \\
\text{chan}: & \text{TransferType} \to \text{Channel} \\
\text{sourceType, destType}: & \text{TransferType} \to \text{Formula} // \text{expressions for conditions} \\
\text{fromMemberGroup, toMemberGroup}: & \text{TransferType} \to \text{Group} \\
\text{amountCond}: & \text{TransferType} \to \text{Formula} \\
\text{custFieldCond}: & \text{TransferType} \times \text{CustFields} \to \text{Formula} \\
\text{where} & \\
\text{Operation} = \{\text{credit, debit}\} \\
\text{Channel} = \{\text{phone, website, pos}\}
\end{align*}
\]

Additional Note:
- ‘Formula’ in the expressions above is a Logic term. In Computer Science it is more often referred to as ‘Boolean-valued expression’. Hence ASM/ASIM formulas can take on (or return) True or False values.

One can imagine $\text{TransferType}$ as a table where each row is named and contains the parameters of the represented $tt$. The $\text{TransferType}$ data type is defined in such a way as to guarantee the following property:

**Transfer Type Welldefinedness.** For each transfer $\text{operation} \in \{\text{credit, debit}\}$, each pair of accounts acc, acc’ and each channel there is at most one $tt \in \text{TransferType}$, i.e. at most one $tt$ which satisfies

- $\text{oper}(tt) = \text{operation}$,
- $\text{chan}(tt) = \text{channel}$,
- $\text{sourceType}(tt) = \text{accountType}(\text{acc})$,
- $\text{destType}(tt) = \text{accountType}(\text{acc'})$.

We denote this unique $tt$ as the value of the function $tt(\text{operation, acc, acc', channel})$.\(^{22}\)

\(^{20}\) The implementation in Cyclos has more parameters we do not consider here.

\(^{21}\) pos abbreviates “point of sale”.

\(^{22}\) Following common notation we use the same name $tt$ for elements of $\text{TransferTable}$ and for the function $tt(\text{params})$; it will always be clear from the context which one is meant.
Both payment operations Credit and Debit are instances of a request/response pattern with two phases, a first phase whose action is called Preview – where only the permission for the transfer is checked (using a \texttt{transferType\_Exstn\_Check} function) but not the requested amount – and a second phase whose action is called Perform where also the amount is checked (using a \texttt{balance\_Check} function). On the user side both actions are treated as stateless, on the INTERLACE network server side only the Preview action is stateless. Here we make the – for the user stateless – two-phase interaction explicit by using two types of user requests, say \texttt{Credit\_Preview\_Req} and \texttt{Credit\_Perform\_Req}, with corresponding Sardex rules \texttt{Credit\_Preview\_Req} and \texttt{Credit\_Perform\_Req} to react to such requests. The user first sends a \texttt{Credit\_Preview\_Req} and upon the positive system response the corresponding \texttt{Credit\_Perform\_Req}. The INTERLACE network server can execute at any moment any of its \texttt{Credit\_Transfer\_Req} rules whose execution depends only on the parameters with which they are called. But for one successful credit request instance the two rules can be executed only in the indicated order, due to the intrinsic sequentiality of the two steps for the request.

\textbf{Remark.} In the \texttt{Credit\_Transfer\_Req} model we assume that its access to the two accounts \texttt{from}, \texttt{to} is exclusive. This assumption plays the role of a constraint for the implementation and thus is an essential part of the specification. We do not model the corresponding synchronization mechanism (which guarantees the assumption) because its functionality is clear and it is well-known how to program it.

\begin{verbatim}
CreditTransferReq((channel, from, to, amount), mbr) =
CreditPreviewReq((channel, from, to, amount), mbr)
CreditPerformReq((channel, from, to, amount), mbr)
\end{verbatim}

We define the two core functions \texttt{transferType\_Exstn\_Check} and \texttt{balance\_Check} of \texttt{Credit\_Transfer\_Req} abstractly, to satisfy a modular design approach, as yielding either a positive answer – for \texttt{transferType\_Exstn\_Check} (in this case by the Transfer Type Welldefinedness property stated above) the matching type \texttt{tt} and for \texttt{balance\_Check} the answer OK – or some information on the reason why the check did not succeed. Since there may be several reasons for a failure, in case of failure the value of the two functions is a set of detected failure reasons, say elements of \texttt{TransferType\_Error} resp. \texttt{Balance\_Violation}. Out of this set an \texttt{ErrMsg} can be constructed to provide the user with some information why the check did not succeed, tailoring the format and payload of error messages. Below we illustrate this modular approach to exception handling by some characteristic examples.

\begin{verbatim}
CreditPreviewReq((channel, from, to, amount, cust\_Flds), mbr) =
if Received(CreditPreviewReq(credit, channel, from, to, amount, cust\_Flds), from: mbr) then
let transfer = (credit, channel, from, to, amount, cust\_Flds)
let ttResult = transferType\_Exstn\_Check(transfer)
if ttResult \notin TransferType then
SEND(ErrMsg(NotPermitted(transfer, ttResult)), to: mbr)
else SEND(YouMayProceedWith(transfer), to: mbr)
CONSUME(CreditPreviewReq(credit, channel, from, to, amount, cust\_Flds), from: mbr)
where
// (Note that the following uses mathematical function notation)
transferType\_Exstn\_Check(transfer) \in
\{ tt \}
if tt \in TransferType and Match(tt, transfer)
\{ PowerSet((TransferType\_Error) else
Match(tt, transfer) if
oper(tt) = credit and chan(tt) = channel and
owner(from) \in from\_Member\_Group(tt) and owner(to) \in to\_Member\_Group(tt) and
source\_Type(tt) = account\_Type(from) and dest\_Type(tt) = account\_Type(to) and
cust\_Field\_Cond(tt, cust\_Flds(transfer)) = true
\end{verbatim}
Additional Notes:

- The above rule should be seen as a specification of a rule, rather than as the implementation of a derived function. Hence, there is no redundancy in the fact that the second line tests whether the CreditPreviewRequest has been received.
- ASM/ASIM functions are mathematical functions that can be discrete (i.e. ‘truth tables’), continuous, static, or dynamic. ASM/ASIM ‘derived functions’ are called ‘subroutines’ or ‘procedures’ in some programming languages.
- ‘∈ PowerSet(TransferTypeError)’ was used instead of ‘∈ TransferTypeError’ because a transfer may violate more than one condition of the Match. So a set of transfer type errors may be returned. If there is a matching tt, then it is unique.
- \{tt\} is a singleton set.

Implementation/refinement comment: the implementation of

```plaintext
Received(CreditPreviewReq(credit, channel, from, to, amount, custFlds), from: mbr)
```

requires refinement. It is of the form

```plaintext
Received(msg; from: s),
```

where the message is

```plaintext
msg = CreditPreviewReq(credit, channel, from, to, amount, custFlds).
```

It is an element of the abstract set ‘Message’, of type ‘credit preview request’ whose content contains values for parameters of type credit, channel, from, to, amount, custFlds. The implementation takes the form

```plaintext
choose m in inboxOf(self) with getMessageSubject(m) = “CreditPreviewRequest” do (etc.)
```

- Specific error cases and their handlers will be defined later. Therefore, there is nothing about this in the present spec.

The Match predicate is extended in Sec. 3.1.3 for the first transfer parameter debit (instead of credit). The set of possible TransferTypeErrors can be defined for the various cases of interest where the Match(tt, transfer) condition is violated for whatever tt ∈ TransferType.

Since the network server when responding to a CreditPreviewRequest does not record the data (due to the requirement of the stateless Preview response character), when elaborating a CreditPerformRequest the system in a first step must redo the transferTypeExstnCheck. In the case of a positive check result, as part of the transaction which is added to the Ledger, besides the transfer parameters also the computed transfer type is recorded together with the transfer date (which is computed by the system as value of a location, say today, when the credit request is performed). The function transaction denotes the final transaction corresponding to the given transfer, its date and transfer type.

To formulate error conditions for the balanceCheck we need a function availBalance(acc) which yields the amount of money that is currently available in the acc to be spent. It is defined together with the related current balance function balance(acc) in Sect. 3.2. Receivable(amount, acc) checks whether the destination of the transfer by receiving the amount would exceed its upper credit limit upperLimit(acc).

---

23 This is the reason why the definition here considers only the if case instead of stating iff.
ConfirmationReq transfer by acknowledging a DebitPerformReq that it uses to interact with the debitor from access to the two accounts

Remark of the three steps for the request instance the three rules can be executed only in the indicated order, due to the intrinsic sequentiality request depends only on the parameters with which they are called. But for one successful debit

The INTERLACE network server can execute at any moment any of these rules whose execution depends only on the parameters with which they are called. But for one successful debit request instance the three rules can be executed only in the indicated order, due to the intrinsic sequentiality of the three steps for the request.

DebitTransferReq = DebitPreviewReq DebitPerformReq DebitAckExec

Additional Note:
- By definition amountCond(ttResult) is a formula. That formula contains a variable, say \( \text{amt} \), which denotes the to-be-checked amount. So \( \text{amountCond}(\text{ttResult})(\text{amount}) \) is that formula with variable \( \text{amt} \) replaced by the actual value \( \text{amount} \). In logic this is also written more formally using ‘/’ for substitution: \( \text{amountCond}(\text{ttResult})(\text{amount}) = \text{amountCond}(\text{ttResult})(\text{amt}/\text{amount}) \).

3.1.3 Behaviour for Debit Operations

In the Sardex business logic also a Debit transfer can be performed but only between accounts of type credit (neither domu nor fee). Since any Sardex member \( c \in \text{SardexNet} \) can be owner of at most one account of type credit, Debit transfers are formulated as being performed among members creditor, debitor. The INTERLACE network server accepts a DebitPreviewRequest and a DebitPerformRequest from a creditor, using two corresponding rules DEBITPREVIEWREQ and DEBITPERFORMREQ that it uses to interact with the creditor. DebitPerformRequests are executed using a request/response interaction between the system and the debitor. The debitor has to allow the transfer by acknowledging a ConfirmationRequest that the INTERLACE network server sends out; only when the debit transfer has been permitted by an acknowledgement from the debitor will the INTERLACE network server execute the transfer using a third rule called DEBITACKEXECUTION.

The INTERLACE network server can execute at any moment any of these rules whose execution depends only on the parameters with which they are called. But for one successful debit request instance the three rules can be executed only in the indicated order, due to the intrinsic sequentiality of the three steps for the request.

Remark. As for CreditTransferReq also in the DebitTransferReq model we assume that the access to the two accounts from, to is exclusive (synchronization constraint).

DebitTransferReq =

DebitPreviewReq
DebitPerformReq
DebitAckExec
Both rules DebitPreviewReq and DebitPerformReq in their first step make a transferType Exstn Check for the account of type credit of the debtor, defined by extending the Match predicate for debit transfer operations. This extension works for B2B operations; it also works for B2E but not for E2B or E2E. In other words Employees are not allowed to request (i.e. appear as first argument of) a Debit transfer.

DebitPreviewReq((debitor, channel, amount, custFlds), creditor) =
  if Received(DebitPreviewReq(debitor, channel, amount, custFlds), from: creditor) then
    let from = creditAccount(creditor), to = creditAccount(debitor)
    let transfer = (debit, channel, from, to, amount, custFlds)
  let ttResult = transferType Exstn Check(transfer)
  if ttResult ≠ TransferType then
    SEND(ErrMsg(NotPermitted(transfer, ttResult)), to: creditor)
  else
    SEND(YouMayProceedWith(transfer), to: creditor)
  CONSUME(DebitPreviewReq(debitor, channel, amount, custFlds), from: creditor)

where
  Match(tt, transfer) if
    oper(tt) = debit and chan(tt) = channel and
    owner(from) ∈ fromMemberGroup(tt) and owner(to) ∈ toMemberGroup(tt) and
    sourceType(tt) = accountType(from) and destType(tt) = accountType(to) and
    custFieldCond(tt, custFlds) = true

If the transferType Exstn Check in a DebitPerformRequest succeeds, a two-phase request/response interaction is started, this time with the system as requester with the debitor to respond. More precisely, upon receiving the DebitPerformRequest from the creditor, the system after a successful transferType Exstn Check executes a balanceCheck, for which we can use the same abstract function as for Credit operations but with interchanged source/destination parameters; in other words, the system checks whether from the creditAccount(debitor) a corresponding Credit operation can be performed. If this check succeeds the system inserts the transaction without further ado if the amount is small (less than 100). Otherwise it creates a OneTimePassword otp, records its birthtime (the beginning of its lifetime), records the otp with the transaction (including the computed transfer type) as pending transaction and sends the otp with an agreement request to the debitor. It then waits for a confirmation.

DebitPerformReq((debitor, channel, amount, custFlds), creditor) =
  if Received(DebitPerformReq(debitor, channel, amount, custFlds), from: creditor) then
    let from = creditAccount(creditor), to = creditAccount(debitor)
    let transfer = (debit, channel, from, to, amount, custFlds)
    let ttResult = transferType Exstn Check(transfer)
    if ttResult ≠ TransferType then
      SEND(ErrMsg(NotPermitted(transfer, ttResult)), to: creditor)
    else
      let bal = balanceCheck(to, from, amount) // check balance
      if bal ≠ ok then SEND(ErrMsg(transfer, bal), to: creditor) else
        if Small(amount) then
          APPEND(transaction(transfer, ttResult, today), Ledger)
          SEND(Confirmed(transfer, ttResult, today), to: creditor)
          SEND(Confirmed(transfer, ttResult, today), to: debitor)
        else
          let otp = new (OneTimePassword)
          birthTime(otp) := now // current system time
          INSERT((otp, transaction(transfer, ttResult)), PendingTransaction)
          status(otp, transaction(transfer, ttResult)) := pending
          SEND(ConfirmationReq(amount, creditor, otp), to: debitor)
        END
      END
      SEND(YouMayProceedWith(transfer), to: creditor)
    END
  CONSUME(DebitPerformReq(debitor, channel, amount, custFlds), from: creditor)
When the \( otp \) is acknowledged (i.e. resent) by the \( \text{debitor} \) within the \( \text{lifetimeForOTPs} \) forseen for one time passwords, the system updates the transaction status from \( \text{pending to performed} \) and \( \text{appends} \) the transaction to the \( \text{Ledger} \) with the current date \( \text{today} \). When the system is waiting for an \( \text{otp} \) confirmation the \( \text{debitor} \) is expected to send, this member may instead try to make another Credit transfer. In this case it could be that only one, Credit or Debit, is still possible due to the \( \text{debitor} \)’s account balance. For this reason, when the pending transaction is confirmed, the \( \text{balanceCheck} \) (but not any more the \( \text{transferTypeExstnCheck} \)) is performed once more and only when it succeeds is the transaction put into the \( \text{Ledger} \). The one time password is deleted to avoid a later application of the rule which has to be applied in case of an \( \text{Expired}(\text{otp}) \).

\[\text{DebitAckExec} =\]
\[\text{if } \text{Received}(\text{DebitAckMsg}(\text{amount}, \text{creditor}, \text{otp}), \text{from}: \text{debitor}) \text{ and not } \text{Expired}(\text{otp}) \text{ then}\]
\[\text{if thereisno } t \in \text{PendingTransaction with } \text{fst}(t) = \text{otp}\]
\[\text{then SEND}(\text{ErrMsg}(\text{IncorrectOtpFor}(\text{amount}, \text{creditor})), \text{to}: \text{debitor})\]
\[\text{else}\]
\[\text{let } t = t'(t' = (\text{otp}, \text{transf}) \mid t' \in \text{PendingTransaction})\]
\[\text{if } \text{status}(t) = \text{pending then}\]
\[\text{let from} = \text{creditAccount}(\text{debitor}), \text{to} = \text{creditAccount}(\text{creditor})\]
\[\text{let } \text{bal} = \text{balanceCheck}(\text{from}, \text{to}, \text{amount})\]
\[\text{if } \text{bal} \neq \text{ok then}\]
\[\text{SEND}(\text{ErrMsg}(\text{transfer}, \text{bal}), \text{to}: \text{creditor})\]
\[\text{SEND}(\text{ErrMsg}(\text{transfer}, \text{bal}), \text{to}: \text{debitor})\]
\[\text{else}\]
\[\text{status}(t) := \text{performed}\]
\[\text{APPEND}(\text{transfer}, \text{today}, \text{Ledger})\]
\[\text{SEND}(\text{Confirmed}(\text{transfer}, \text{today}), \text{to}: \text{creditor})\]
\[\text{SEND}(\text{Confirmed}(\text{transfer}, \text{today}), \text{to}: \text{debitor})\]
\[\text{DELETE}(\text{otp, OneTimePassword})\]
\[\text{CONSUME}(\text{DebitAckMsg}(\text{amount}, \text{creditor}, \text{otp}), \text{from}: \text{debitor})\]
\[\text{where}\]
\[\text{Small}(\text{amount}) \text{ iff } \text{amount} < 100\]
\[\text{Expired}(\text{otp}) \text{ iff } \text{now} - \text{birthtime}(\text{otp}) > \text{lifetimeForOTPs}\]

In case the \( \text{debitor} \) does not confirm the Debit request within the \( \text{lifetime} \) forseen for OTPs, the INTERLACE network server will reject the \( \text{DebitPerformReq} \) (by changing its status to \( \text{rejected} \)) and inform the \( \text{creditor} \) about it.

\[\text{DebitRejectExec} =\]
\[\text{if } \text{otp} \in \text{OneTimePassword and Expired}(\text{otp}) \text{ then}\]
\[\text{if thereisno } t \in \text{PendingTransaction with } \text{fst}(t) = \text{otp}\]
\[\text{then SEND}(\text{ErrMsg}(\text{IncorrectOtpFor}(\text{amount}, \text{creditor})), \text{to}: \text{debitor})\]
\[\text{else}\]
\[\text{let } t = t'(t' = (\text{otp}, \text{transf}) \mid t' \in \text{PendingTransaction})\]
\[\text{if } \text{status}(t) = \text{pending then}\]
\[\text{status}(t) := \text{rejected}\]
\[\text{SEND}(\text{ConfirmRejectMsg}(\text{Rejected}(\text{amount}, \text{creditor})), \text{to}: \text{debitor})\]
\[\text{DELETE}(\text{otp, OneTimePassword})\]

Remark. Up to now request/response pattern time issues are not formulated in the model. Here this concerns in particular the timeout mechanism for pending transactions.\(^{26}\)

\(^{24}\) \( \text{fst} \) denotes the first element of a sequence, here of a pair \( (\text{otp}, t) \) of an \( \text{otp} \) and the corresponding pending transaction \( t \).

\(^{25}\) Hilbert’s \( \epsilon \)-operator \( \text{exP}(x) \) denotes the unique \( x \) which satisfies property \( P \).

\(^{26}\) I can add such a mechanism once the rules become sort of stable.
3.2 Account History and Balance Operations

The INTERLACE system accepts from every business member $c \in \text{SardexNet}$ an account history request for any of its accounts, i.e. the elements of the set $\text{Account}(c)$. As parameters of such a request the member can indicate besides the account also the period $\in \text{Period}$ for which the history is requested, the counterparty (an element of the set $\text{CounterParty}(c)$ of accounts allowed to be accessed by the member for a transfer), and/or the amount range in the set $\text{AmountRange}$ of allowed transfer amounts, as well as some custom fields (elements of the set $\text{CustField}$).

As usual in the model the information requested by the parameters is retrieved by applying corresponding functions to transactions. A history request is about transactions $t$ in the Ledger, where the account appears as $\text{source}(t)$ – the account from where the $t$-transfer has been made – or as $\text{dest}(t)$, the account where the $t$-transfer has been directed to; here $\text{source}$ and $\text{dest}$ indicate the corresponding extraction functions applied to transactions, formally:

$$
\text{source} : \text{Transaction} \rightarrow \text{Account} \\
\text{dest} : \text{Transaction} \rightarrow \text{Account} \\
\text{date} : \text{Transaction} \rightarrow \text{Time} \\
\text{amount} : \text{Transaction} \rightarrow \text{Amount} \\
\text{customFields} : \text{Transaction} \rightarrow \text{CustomFields} \\
\text{counterParty} : \text{Transaction} \times \text{Account} \rightarrow \text{PowerSet}(\text{SardexNet}) \\
\text{transferType} : \text{Transaction} \rightarrow \text{TransferType}
$$

where

$$
\text{Amount} = \{ n \cdot x y \mid n \in \text{Nat} \text{ and } 0 \leq x, y \leq 9 \} \\
\text{CustomFields} \subseteq \text{CustField}
$$

For to-be-reported transactions $\text{date}(t)$ must be within the indicated period. The $\text{counterParty}(t, \text{acc})$ function extracts from a transaction $t$ the owner of the other account involved in the transaction and is applied in case the $\text{counterPty}$ parameter is not $\text{All}$. The $\text{amount}(t)$ is required to be in the indicated $\text{AmountRange}$. A $\text{CustFldMatch}$ condition expresses the relation which is requested to hold between the $\text{custField}$ parameter and the custom fields extracted by the function $\text{customFields}(t)$.

The INTERLACE network server answers an $\text{AccountHistReq}$ by sending back to the requestor either an error message – in case the requestor is not a member of the circuit or the indicated account is not one of its accounts – or the set $T$ of transactions which satisfy the above-indicated properties. This is specified by the following ASM rule.

$$
\text{ACCOUNTHistReq} = \\
\text{if } \text{Received}(\text{AccountHistReq}(\text{acc, period, counterPty, amountRange, custFld}), \text{from: } c) \text{ then}
$$

$$
\text{if } c \notin \text{SardexNet} \text{ or } \text{acc} \notin \text{Account}(c)
$$

$$
\text{then } \text{SEND}(\text{ErrMsg}(\text{youHaveNoSuch}(\text{acc})), \text{to: } c)
$$

$$
\text{else}
$$

$$
\text{let } T = \{ t \in \text{Ledger} \mid (\text{source}(t) = \text{acc} \text{ or } \text{dest}(t) = \text{acc})
\text{ and } \text{date}(t) \in \text{period} \text{ and } \text{amount}(t) \in \text{amountRange}
\text{ and (if } \text{counterPty} \neq \text{All then } \text{counterParty}(t, \text{acc}) \in \text{counterPty})
\text{ and } \text{CustFldMatch}(\text{custFld}, \text{customField}(t)) \} \text{ in}
$$

$$
\text{SEND}(T, \text{to: } c)
$$

$$
\text{CONSUME}(\text{AccountHistReq}(\text{acc, period, counterPty, amountRange, custField}))
$$

In a similar way, one can specify the behaviour of the INTERLACE network server when it receives a $\text{BalanceRequest}$, namely to compute the current $\text{balance}$ of the requestor’s account. This computation calculates the sum of the amounts of each received transfer and detracts from it the sum of the amounts of each sent transfer.
In addition, we foresee that, for performance and database management reasons, from time to time the system issues a certifiedBalance. Therefore to calculate the current balance(\textit{acc}), starting with the last certifiedBalance of this account, only those transactions need to be considered whose date is after the last balanceCertificationDate(\textit{acc}), a dynamic location the system updates to the system time \textit{now} each time it updates the value of the location certifiedBal(\textit{acc}). This is expressed by the following ASM rule:

\[
\text{BalanceReq} =
\begin{align*}
\text{if } & \ \text{Received(BalanceReq(}\textit{acc}), \text{from: mbr)} \text{ then} \\
& \text{if } \ mbr \not\in \text{SardexNet} \text{ or } \textit{acc} \not\in \text{Account}(\textit{mbr}) \\
& \text{then } \text{SEND}(\text{ErrMsg(youHaveNoSuch(\textit{acc}})), \text{to: mbr)} \\
& \text{else} \\
& \text{let } \text{In} = \{ t \in \text{Ledger} | \text{dest}(t) = \textit{acc} \text{ and } \text{date}(t) > \text{balCertificationDate(\textit{acc}}) \} \\
& \text{// case receive} \\
& \text{let } \text{Out} = \{ t \in \text{Ledger} | \text{source}(t) = \textit{acc} \text{ and } \text{date}(t) > \text{balCertificationDate(\textit{acc}}) \} \\
& \text{// case transfer} \\
& \text{let } \text{bal} = \sum_{t \in \text{In}} \text{amount}(t) - \sum_{t \in \text{Out}} \text{amount}(t) + \text{certifiedBal(\textit{acc}}) \\
& \text{let } \text{NoOfTransactions} = | \text{In} | + | \text{Out} | \\
& \text{SEND}((\text{bal}, \text{NoOfTransactions}, \text{to: mbr)} \\
& \text{CONSUME(BalanceReq(\textit{acc}}))} \\
\end{align*}
\]

Having the balance, one can compute the availBalance (the spendable amount) by adding the value of the creditLine:

\[
\text{availBalance(\textit{acc}}) = \text{balance(\textit{acc}}) + \text{creditLine(\textit{acc}})
\]

\text{availBalance} is an example of a derived function, i.e. a dynamic function with a fixed computation scheme (here an equation). \text{creditLine(\textit{acc}}) is a monitored function for members, it is a controlled function for the agent (typically a broker) who has the right to set it.

### 3.3 User Operations

Users can SEND requests which appear as input for the INTERLACE network server. To SEND(\text{CreditPreviewReq(\textit{transfer}})) or to SEND(\text{DebitPreviewReq(\textit{transfer}})) is conditioned only by a correct definition of the \textit{transfer} parameter, definition the user supplies by filling in the corresponding fields on the screen. The same holds mutatis mutandis for SEND(\text{AccountHistReq(\textit{histParams}}) and SEND(\text{BalanceReq(\textit{acc}})). The functionality is clear so that we do not model further this editing process.

For Credit/Debit Perform requests the only relevant additional constraint is that they can be sent only after an ok-message for the corresponding Preview request has been received. We use a function \textit{kind} to extract from a \textit{transfer} parameter its credit resp. debit component.\textsuperscript{27}

\[
\text{if } \ \text{Received(YouMayProceedWith(\textit{transfer}}, \text{from: sardex)} \text{ then} \\
& \text{if } \ \text{kind(\textit{transfer}}) = \text{credit} \text{ then} \\
& \text{SEND(\text{CreditPerformReq(\textit{transfer}}, \text{to: sardex)} \\
& \text{if } \ \text{kind(\textit{transfer}}) = \text{debit} \text{ then} \\
& \text{SEND(\text{DebitPerformReq(\textit{transfer}}, \text{to: sardex)} \\
& \text{CONSUME(YouMayProceedWith(\textit{transfer}}))}
\]

\textsuperscript{27} In the following ASMs the keyword ‘sardex’ stands for ‘INTERLACE network server’.
In case of a Debit operation a debitor has to confirm a received debit request by sending a DebitAckMsg; otherwise a DebitRejectMsg is sent to the INTERLACE network server.

if Received(ConfirmationReq(amount, creditor, otp), from: sardex) then
    if Agreed(amount, creditor, otp)
        then Send(DebitAckMsg(amount, creditor, otp), to: sardex)
    else Send(DebitRejectMsg(amount, creditor, otp), to: sardex)
    Consume(ConfirmationReq(amount, creditor, otp))
References


