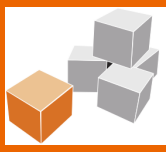
Local-First Enterprise Applications

Master thesis by André Wolski
Date: 17.04.2024

1. Review: Prof. Dr.-Ing. Mira Mezini
2. Review: Dr.-Ing. Ragnar Mogk
Darmstadt

TECHNISCHE UNIVERSITÄT DARMSTADT



Software Technology Group
TU Darmstadt | FB Informatik

Computer Science Department
Software Technology Group

Local-First
Enterprise Applications

Master thesis by André Wolski

Date: 17.04.2024

Darmstadt

Bitte zitieren Sie dieses Dokument als:
URN: urn:nbn:de:tuda-tuprints-270064
URL: http://tuprints.ulb.tu-darmstadt.de/27006
Jahr der Veröffentlichung auf TUprints: 2024

Dieses Dokument wird bereitgestellt von tuprints,
E-Publishing-Service der TU Darmstadt
http://tuprints.ulb.tu-darmstadt.de
tuprints@ulb.tu-darmstadt.de

Erklärung zur Abschlussarbeit gemäß § 22 Abs. 7 APB TU Darmstadt

Hiermit erkläre ich, André Wolski, dass ich die vorliegende Arbeit gemäß § 22 Abs. 7 APB der TU Darmstadt selbstständig, ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt habe. Ich habe mit Ausnahme der zitierten Literatur und anderer in der Arbeit genannter Quellen keine fremden Hilfsmittel benutzt. Die von mir bei der Anfertigung dieser wissenschaftlichen Arbeit wörtlich oder inhaltlich benutzte Literatur und alle anderen Quellen habe ich im Text deutlich gekennzeichnet und gesondert aufgeführt. Dies gilt auch für Quellen oder Hilfsmittel aus dem Internet.

Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Mir ist bekannt, dass im Falle eines Plagiats (§ 38 Abs. 2 APB) ein Täuschungsversuch vorliegt, der dazu führt, dass die Arbeit mit 5,0 bewertet und damit ein Prüfungsversuch verbraucht wird. Abschlussarbeiten dürfen nur einmal wiederholt werden.

Bei einer Thesis des Fachbereichs Architektur entspricht die eingereichte elektronische Fassung dem vorgestellten Modell und den vorgelegten Plänen.

Darmstadt,

André Wolski

Abstract

Local-First Software has been proposed in 2019 by Kleppmann et al. [11] to address concerns with the growing number of cloud-only web-applications, as these move control from the end users to the cloud providers. Local-First Software stores all necessary data locally, and has the cloud only as an optional utility, to support cross-device synchronization and collaboration with other users.

Enterprise Applications are traditionally implemented with a client-server architecture, and are nowadays also following the trend of cloud-only web-applications. The move to the cloud raises similar concerns about vendor lock-in, resilience and business continuity, data protection, and privacy.

In this master thesis, we investigate if the ideals of Local-First Software can be applied to Enterprise Applications. We look at three business processes to understand the business requirements for Enterprise Applications. We then discuss how these match with the ideals of Local-First Software, and propose 12 requirements for Local-First Enterprise Applications. We will see that these still require centralized systems, albeit with a weaker dependency than traditional Enterprise Applications.

We develop a prototype to further analyze and discuss if and how Enterprise Applications can be developed based on the ideals of Local-First Software. For this, we develop two underlying libraries as a foundation for connection management and CRDT state replication over a hierarchical peer-to-peer network.

Contents

1 Introduction

Enterprise Applications are software products, that are tailored to the needs of an *Enterprise*. Enterprise Applications are essential to perform standard business processes, for example with *Enterprise Resource Planning (ERP)*, *Customer Relationship Management (CRM)*, or *Supply Chain Management (SCM)* systems, typically available as *off-the-shelf applications*. Enterprise Applications also cover specialized applications that are developed for a specific use-case, i.e., a non-standard business process, and for a single customer, so called *bespoke applications*.

Challenges in the development of Enterprise Applications are complex business requirements, high standards for legal compliance, inter-dependencies between different Enterprise Applications, and their scale – with sometimes several thousand active users at a time. Historically, Enterprise Applications have been developed originally as mainframe/terminal applications, evolved to client-server desktop-applications, and are nowadays following the trend of cloud-based web-applications. The architecture of Enterprise Applications, called enterprise architecture, is typically following a design where end devices have a limited local state and control, and are usually requiring a constant network connection to a central system (central in a logical sense – this ranges from a single physical server to a highly distributed microservice architecture with several geo-distributed endpoints).

With the trend of cloud-based web applications increases the dependency on and power of cloud (application) providers, which raises the risk of a vendor lock-in, and by giving control over the data to the vendor, longevity and resilience are threatened – a vendor might discontinue a service that is not profitable or no longer part of their strategy, whereas the Enterprise depends on the service for critical business processes, and also has to keep, for example, financial records available for several years.

Motivated by the concerns related to cloud-based applications, ***Local-First Software*** has been proposed in Kleppmann et al. [11] by presenting seven ideals for an alternative application design, which gives more control and data to end users and their local devices. With data available locally, a network connection is optional, which enables users to use

the application offline and have a more responsive user experience. With the cloud as an optional component and not a requirement, the control over the data is given back to the users, which also ensures longevity of the data, and improves security & privacy. Given a network connection, the data can be synchronized automatically and is available on multiple devices, which also enables collaboration with other users.

The seven ideals for Local-First Software are focused on applications that are either document-based, like text, graphic, sound, or design editors, or have a flat data structure, like a to-do list or a calendar. The paper however explicitly excludes more complex or interconnected applications like “banking software”, “e-commerce” – or more general speaking: Enterprise Applications.

In this master thesis, we will explore if and how the requirements for Enterprise Applications can be matched with the ideals of Local-First Software, i.e., if these two can be combined to ***Local-First Enterprise Applications***. In the next section of this introduction we will look Enterprise Applications and their architectures. In the next Chapter 2 on page 6 we will introduce the fictional company *Aperture Science Laboratories Inc.* as an abstracted exemplary company, which we will then use to investigate requirements for Local-First Enterprise Applications. In Chapter 3 on page 27 we will solve some of the open requirements, as we will develop a library for connection management within ScalaLoci, a library for CRDT state replication within REScala, and an example application to demonstrate a) how these libraries work & interact and b) how a Local-First Enterprise Application could be constructed. In Chapter 4 on page 43 we will evaluate the performance of the replication and compare the achieved results with the requirements. In Chapter 5 on page 48 we will have a summary and conclusion.

1.1 Enterprise Applications

Following the origin of the word, an *Enterprise* is a form of an activity, that has a specific goal. In the original sense, it is synonymous to an endeavor or a venture, and an enterprise could be an expedition or a research & development project. Nowadays however, the term ***Enterprise*** is used to describe organizations that have a minimal size of members or employees, to distinguish “enterprise organizations” from *Small Office, Home Office (SOHO)* organizations and from individuals. Although there is a definition of *Small and Medium-sized Enterprises (SMEs)* in the EU [2] for companies up to 50 resp. 250 employees, which would imply those are also enterprises, the colloquial usage of the standalone term

Enterprise refers to larger companies and corporations with more than 250 employees, which we will follow in this thesis.

Typically, Enterprises follow a common structure, by organizing their supporting functions into departments, like a Finance department, an IT department, or Marketing department. Larger Enterprises may additionally be structured by region; in practice, commercial enterprises often have at least one subsidiary in each country/legislation they operate in, to align with the regional regulations, e.g. a GmbH in Germany and an SARL in France. Some departments, like the finance department, may be present in each subsidiary, while others, like the IT department, may only be suited once on the top-level entity.

Enterprise Applications are used to support the processes that occur in each department, with a varying degree of integration with other Enterprise Applications from other departments or entities. For example, each Finance department from each country could have their own *Enterprise Resource Planning (ERP)* system, or they could share a common global instance. Likewise, the Marketing department could have a standalone *Customer Relationship Management (CRM)* system, or have it integrated as one application with the ERP system.

The technical design of each Enterprise Application and their composition is concern of the *Enterprise Architecture* field. Traditional Enterprise Applications are developed with a desktop application, that is connected over a network to one or many application servers, which store their data in one or many database servers. This is depicted in the first row of Figure 1.1 on the right side.

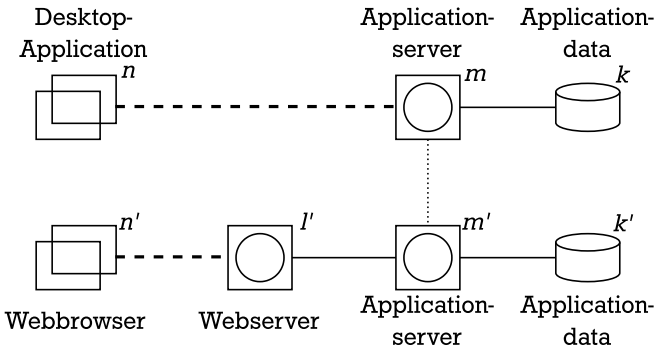


Figure 1.1: Two interdependent Enterprise Applications

The second row of that figure shows the basic design of more recent Enterprise Applications: instead of having a dedicated desktop application, the application is used in a web browser, which loads the UI and state from a web server. This figure also shows that Enterprise Applications are often interconnected; they may exchange some state or are performing operations on each other.

This figure uses the notation as it is introduced in the book *Moderne Enterprise-Architekturen* by Masak [12], which presents and analyses enterprise architectures and their topologies.

With today's globalization, Enterprise Applications are also scaled on a global level, often being used daily by every employee from any location across the globe. A challenge in the design of any distributed application is the necessary tradeoff between consistency, availability, and performance.

1.2 Business Process Management

Business Process Management (BPM) is used to organize reoccurring activities in an organization, by describing the conditions and steps of an activity as a process. The extend of which BPM is used within an organization can vary, some processes may only be described informally and document a high-level ideal of how the process should perform, other processes may be analyzed and documented in detail, for example, with a graphical notation from the *Business Process Model and Notation (BPMN)* or from *Subject-oriented Business Process Management (S-BPM)*.

A general introduction into Business Process Management (German: *Geschäftsprozessmanagement*) is available in the book *Geschäftsprozessmanagement in der Praxis: Kunden zufriedenstellen, Produktivität steigern, Wert erhöhen* by Hermann J. Schmelzer and Wolfgang Sesselmann [13] from 2013. A deeper perspective in how business processes can be supported by IT systems is provided in the book *Ganzheitliche Digitalisierung von Prozessen* by Albert Fleischmann et al. [10] from 2018, which also compares different modeling notations in detail.

In this thesis we will use the S-BPM methodology, which is centered around so-called *Subjects* as active entities in a process, that perform tasks and exchange messages between them. Depending on the chosen level of detail for the process model, a Subject may represent humans, a complete department, a specific role within a department, or a technical system, like an active database or even a single microprocessor or sensor.

The level of detail for a process model is selected depending on the intended audience; a process can also be described from multiple views with different process models. In Figure 1.2 a typical segmentation into three levels is shown. First, a high-level process model is created on the business level with domain experts, which gathers the major steps of the business logic and involved subjects. On this level, the involved subjects are typically employees with a certain role or from a specific department; if necessary for the business logic, there can also be technical subjects, for example, if it is known that some parts of the business logic must not or cannot be executed by a human.

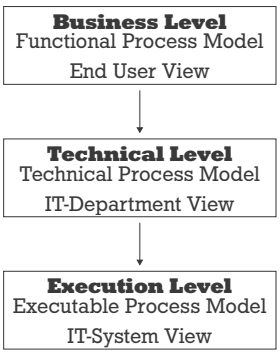


Figure 1.2: Modeling Levels, adapted from Abb. 10.7 in [13]

For the development of Enterprise Applications, the process models from the business level are used as a reference and are refined into more detailed process models on the technical level, which will contain technical details and implementation decisions. For example, technical subjects are introduced to represent the necessary data sources & storages and communications with other IT systems and Enterprise Applications.

The implementation of a business process can done with two exclusive or complementary directions: via software development as code and/or with a fine-grained executable process model, that is interpreted by an execution engine, which is usually available as part of an ERP system.

The business process models that will be presented in this thesis are situated on the business level and will contain some technical subjects if needed. For the scope of this thesis, the *Parallel Activity Specification Schema (PASS)* of the S-BPM methodology is used, as it is presented in [15, 16, 17]. Only the high-level *Subject Interaction Diagram (SID)* is relevant and the more detailed *Subject Behavior Diagram (SBD)* will not be used in this thesis.

2 Identifying Requirements & Ideals

Throughout this work, we will look at the fictional company *Aperture Science Laboratories Inc.* as an abstracted exemplary company, which represents aspects of real-world companies, so that examples and requirements are easy to demonstrate and are not too generalized. We construct Aperture Science Laboratories Inc. as a company, that is interested to invest in Local-First Software for their own internal usage, which is motivated by some unique properties that sets them apart from typical office-based corporations.

Aperture Science Laboratories Inc. is a globally operating research and development corporation with 50.000 employees, providing their know-how on advanced technologies to other companies and government agencies. Overall, they have about 500 offices, research & development centers, laboratories, and research outposts. Their global headquarter has 5.000 employees and is located in a major city in North America. Their German headquarter is located in Darmstadt, Hesse with an office for 500 employees.

Aperture Science Laboratories Inc. is interested in a good public image and efficient internal processes, to be an attractive employer and to win profitable projects without much overhead in their administrative functions. To continue their good relationship with government agencies, they are very keen to comply with any legal requirements. Legally, Aperture Science Laboratories Inc. is structured as a corporate group, with at least one daughter company in each country they are operating in, leading to heterogeneous legal frameworks to comply with.

As many projects are classified, using public clouds and external vendors is challenging. On the other hand – to maintain productivity – are employees allowed to work on unclassified projects from home or while traveling. This unique combination motivated the IT department of Aperture Science Laboratories Inc. to review their existing application portfolio, which not only contains typical commercial off-the-shelf software but also several bespoke applications, of which some are externally and others internally developed. Thereby, they are reviewing their IT architecture and are investigating Local-First Software as an alternative concept.

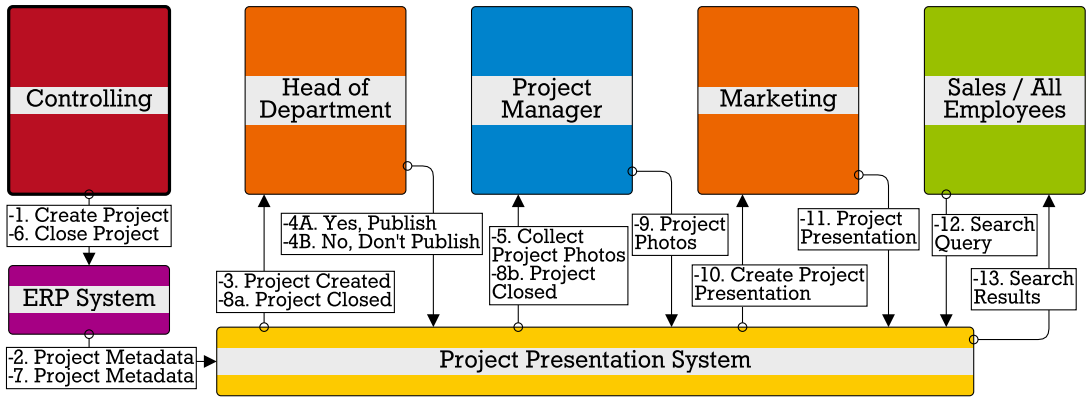


Figure 2.1: Process Model: Create Project Presentation

2.1 Business Process: Create Project Presentation

When the sales department approaches a new potential client, submits a bid on an *Invitation to Tender (ITT)* or a proposal on a *Request for Proposal (RFP)*, they want to demonstrate that Aperture Science Laboratories Inc. is a competent partner and has experience in the area of interest of the client and with the expected work.

For this, the Sales department is using a bespoke application to create and manage marketing presentations of selected successfully completed non-confidential projects, which highlight the performed work in each project. To always have the latest information about all projects and to not miss any new project, this Project Presentation System is connected to the Enterprise Resource Planning (ERP) system and automatically receives any changed metadata of each project.

Happy Path Description

The process to fill and maintain this database is shown on a high-level as a Subject-Interaction-Diagram (SID) in Figure 2.1. The process starts in the controlling team of the finance department, when a new project is created and entered (1.) into the ERP system. For each project, the ERP system stores the project number, status, client, and confidentiality level. This data is transmitted from the ERP system to the Project

Presentation System (2.). Once the project is available in the Project Presentation System, the head of the corresponding department (HoD) will receive a notification (3.) about this new project. Given the project classification and considering the effect on the public image, the HoD may decide to request the creation of a presentation file (4A.) or not (4B.).

The corresponding Project Manager (PM) is informed that interesting photos from the project should be made (5.), which will later be used in the presentation file. Once the project is completed, the PM will receive a notification (8b.) to review and submit the collected photos (9.). Once the photos of the project are submitted, and if a presentation file needs to be created, the system will send a notification (10.) to the marketing department. They will create a presentation file out of the available metadata and photos and upload it to the system (11.).

At the end of the process, the sales department can query (12.) the database and receive (13.) all details about the project. In fact, all employees are able to submit queries to the database – however, the search results have to respect the project classification and the employee's role.

Variations and Exceptions

Deviating from the shown happy path, variants and exceptions have to be considered. This very high level description serves as an entry point to be further refined, in case development of a new implementation is commenced. Optional steps has been left out of the happy path, to reduce the complexity of the presented process model.

As a variation from the happy path, the Head of Department can revert the decision to create a presentation or not, for example, when a project that seemed irrelevant becomes a good candidate, or if the classification level changes during the project execution. The Project Manager may upload and delete photos as long as the presentation has not been finalized by the Marketing department.

As an exception, members of the Sales department may act on-behalf of the Head of Department or the Project Manager, for example in cases where a change needs to be made ad-hoc if a person is unavailable or cannot access the system. This is usually justified based on external information or approvals and needs to be documented.

Business Requirements

Business Requirement 1.1: ERP Integration

The application needs to be able to receive data from other applications, like an Enterprise Resource Planning (ERP) system. This is not time critical and could be pull-based.

Business Requirement 1.2: Notifications

The application needs a mechanism to notify employees about new tasks. This may be a system notification when the application is running in background, or may be email based. For the Head of Department and the Project Manager, this is not time critical and may be a delayed email that aggregates several notifications in a daily summary.

Business Requirement 1.3: Tasklist

User who have outstanding tasks, for example a Head of Department who has to decide if some projects need a presentation or not, need to see all their outstanding tasks with a way to navigate to them.

Business Requirement 1.4: Group-based Permission Management

Employees of both the Sales and the Marketing department need general access to the application. This may be implemented as part of the application with a user management system, or may be dynamically connected to Active Directory groups.

Business Requirement 1.5: Entry-based Permission Management

The Head of Department and the Project Manager differ in each project and have to be derived from the ERP data. A Head of Department can only make decisions for projects of their department, a Project Manager can only manage photos of their own projects.

Business Requirement 1.6: State-based Permission Management

Only after publication of the final presentation are All Employees able to see the entry and the presentation file. Entries, photos and files of confidential projects are not visible for All Employees.

Business Requirement 1.7: Document Storage

The application needs to be able to handle many large files, like photos and the final presentation. The application may store the files in its own database, or may be integrated with an external storage or Document Management System (DMS).

2.2 Business Process: Incident Report

Like any other company, Aperture Science Laboratories Inc. is responsible for the well-being of their employees and visitors. This responsibility is given to the Health & Safety department; on a global level to steer and on each country to perform and align with local regulation. A key business process, that the global H&S department has developed as a scheme for all daughter companies, is the "Incident Report" process, which standardizes how the reporting of incidents, that affect the health or the safety of employees or visitors, is done.

The primary goal of the "Incident Report" process is to comply with legal requirements. For some incidents, local authorities may need to be notified within a certain deadline. The definition of which incidents need to be reported and the respective deadlines depends on local regulations. For example, in Germany an incident needs to be reported to a *Berufsgenossenschaft (BG)* (employer's liability insurance), if an insured person is unable to work for three or more calendar days due to the incident, within three calendar days after receiving word of the incident [7]. In the US, work related incidents need to be reported to the *Occupational Safety and Health Administration (OSHA)* within 7 calendar days [4].

A secondary goal of the "Incident Report" process is to investigate the cause of the incident, to avoid similar incidents in the future, for example, by updating work instructions and improving H&S training materials. Reporting on regular intervals, aggregated across all daughter companies, shows to the management trends and highlights which areas need improvement the most.

Due to the varying local regulations, each country Aperture Science Laboratories Inc. operates in has developed their own country-specific application. The global H&S leadership wants to rationalize their IT portfolio and welcomes the IT department's investigation of Local-First Enterprise Applications as an opportunity to harmonize the Incident Report process globally.

Happy Path Description

The "Incident Report" process, shown in Figure 2.2 on the facing page, starts, when a reporter fills out and submits (1.) an initial incident report. The initial incident report contains a preliminary classification about the severity of the incident, which determines

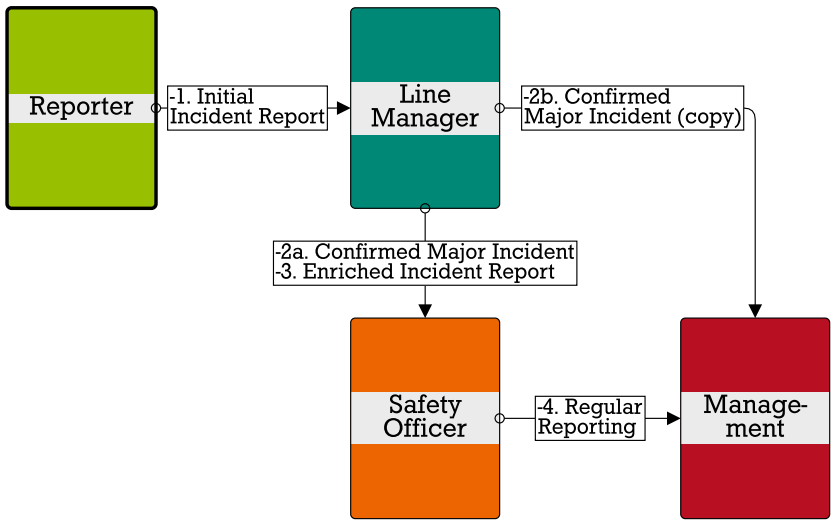


Figure 2.2: Process Model: Incident Report

the deadlines for notification of the authorities. The reporter can be any employee of Aperture Science Laboratories Inc., and is typically affected by the incident or witnessed it.

To ensure 1. that the proper deadlines are met and 2. the incident report contains all relevant information, the Line Manager (LM) of the reporter performs two subsequent reviews. In the first review, the Line Manager confirms or updates the severity of the incident; in case of an incident with major severity a notification is send to the Safety Officer (2a.) and to the Management team (2b.), who will arrange the proper notification of the respective local authority.

In the second review, the Line Manager is responsible to further document the incident and send an enriched incident report (3.) to the Safety Officer. The Safety Officer will periodically aggregate all received enriched incident reports and prepare a compact report (4.) to the Management, which summarizes all incident reports of the period, and highlights suggestions to avoid future incidents.

Variations and Exceptions

To meet the legally mandated ‘deadlines, when an initial incident report does not receive the first review within a certain time, a reminder will be send to the Line Manager and also to the Safety Officer. The Safety Officer can step in at any time for any action of the Line Manager, especially to perform the first review, but also to complete the documentation and enriched incident report.

Mandatory Business Requirements

Business Requirement 2.1: Notifications

The application needs a mechanism to notify employees about new tasks. In case of severe incidents, this is time critical and needs to be reliable.

Business Requirement 2.2: Entry-based roles

The Line Manager is determined via the Reporter. The Management is determined via the region, e.g., per country. The Safety Officer is determined via the location, e.g., the office or facility.

Business Requirement 2.3: Data Protection

Incident Reports contain sensitive personal information, which are subject to a higher level of protection under GDPR Article 9 [3]. Incident Reports, or at least the sensitive part of a report, must only be accessible on a need-to-know basis, i.e., the Reporter, their Line Manager, their Safety Officer and in case of severe incidents the Management. For reporting and analysis, reports must be sufficiently redacted, at least but not only exclude the name and photos.

Business Requirement 2.4a: Regional Variations

The application needs to be customizable per region. Different deadlines for notifications per country are a must-have.

Business Requirement 2.5: Reporting

To identify trends, the Safety Officers need to categorize, filter and visualize the number of reports, for example, a graph of trip hazard related incidents over time and per location.

Business Requirement 2.6: Document Storage

The application needs to be able to handle many large files, like photos of the

incident location. The application may store the files in its own database, or may be integrated with an external storage or Document Management System (DMS).

Optional Business Requirements

Business Requirement 2.4b: Regional Variations

Different regions want to have different fields for the incident report, as they have to provide different information to the authorities when reporting severe incidents. Custom notification templates are nice-to-have, so that these can refer to the regional procedure for reporting incidents to the authorities.

Business Requirement 2.7: Offline Creation

Ideally, the initial incident report can be prepared without a network connection, as incidents occur not only in location with good network availability like offices, but also during travel or in remote facilities with limited or no network connectivity.

Business Requirement 2.8: ERP Integration

The incident report may contain optional fields to link an incident to an ongoing project, to identify and report project or client related trends.

2.3 Business Process: Procure-to-Pay

With the Procure-to-Pay process, we will look at one of the standard core processes of traditional Enterprise Resource Planning (ERP) systems. This business process is owned by the Finance department and involves their Procurement and their Accounts Payable (AP) team. It is the major way to spend money, from ordering small physical goods like personal protective equipment (PPE) to long-running & high-volume service contracts.

From an economic perspective, the finance department has to ensure that the enterprise has sufficient cash at any given time, to fulfill any financial obligation in time – otherwise it would be considered as insolvent, even though they might have sufficient other current assets and fixed assets. From a legal perspective, the finance department needs to have a correct accounting at all times, needs to document every transaction, and has to provide regular financial statements. From a management perspective, the finance department has to ensure, that every spend is authorized.

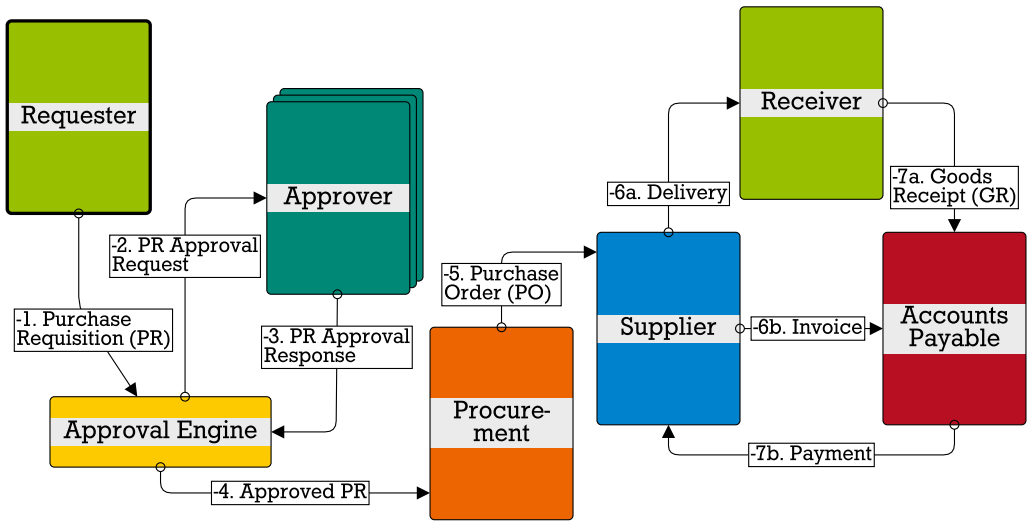


Figure 2.3: Process Model: Procure-to-Pay

The Procure-to-Pay process supports these high-level requirements, by channeling all orders through an review and authorization process, thereby documenting all commitments and giving an overview of upcoming payments.

Happy Path Description

The process model of the Procure-to-Pay process is shown on a high-level as a Subject-Interaction-Diagram (SID) in Figure 2.3.

The process starts when a Requester enters a new Purchase Requisition (PR) (1.), which contains a description or reference of goods or services that are needed, their expected price, the desired supplier, the expected payment terms, the desired delivery information and receiver, and an assignment to a project or department which will cover the expense. Based on the assigned project or department and the expected price, the Approval Engine will select one or more approvers and sends them a notification (2.) which they will acknowledge or reject (3.). Once the Approval Engine received all the necessary approvals, it will forward the approved PR (4.) to the Procurement team. A member of the Procurement team will review the request and thereby may change the supplier, delivery information, expected price, or payment terms, for example, after checking the availability

on the supplier side. Multiple PRs from different Requesters may be combined into a single order, to reduce the number of deliveries. The last step of the Procurement team is to finalize and then send a legally binding Purchase Order (PO) (5.) to the supplier.

The supplier will deliver the goods or services (6a.) to the receiver, that is stated in the delivery information, who then will validate and confirm the proper fulfillment of the order, and document this by sending a Goods Receipt (GR) (7a.) to the Accounts Payable (AP) team.

The supplier will also send an Invoice (6b.) to the Account Payable (AP) team; depending on the payment terms, this may happen before, at the same time, or after fulfilling the order. Once the AP team receives the Invoice, a member of the team reviews if the invoice matches the order and the agreed payment terms.

When the GR happens before the AP team received and booked the Invoice, then the system will automatically account an accrual; once the Invoice is entered, the payment will be initiated (7b.) and booked against the accrual. If however the Invoice arrived before a GR, the behavior depends on the payment terms agreed in the PO. If a prepayment has been agreed, the payment will be initiated (7b.). Otherwise, the payment is held back until the GR is received as a confirmation of the fulfilled order, and a liability is booked.

Related Processes

The Procure-to-Pay process interacts with many different business processes and ERP modules. Purchase Requisitions (PRs) can only be created for a project that is active; on the other hand, a project cannot be closed as long as not all Purchase Orders (POs) have been completed. Another interaction with the Project Management module is that the Approval Engine uses the Project Manager as approver. To reduce

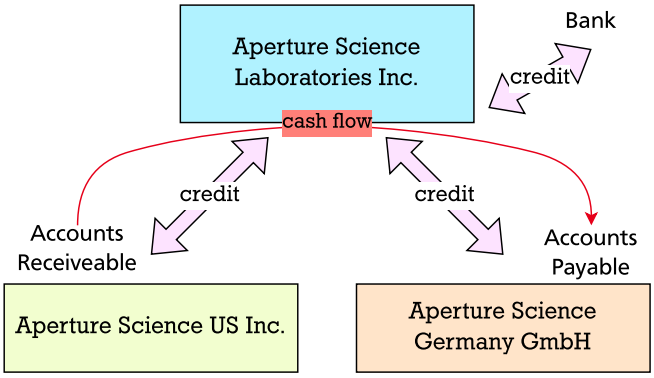


Figure 2.4: Cash-Management, adapted from Bild 2 in [14]

the number of exceptional approvals, the Approval Engine is connected to the Human Resource Management module and follows absence delegations. Further, Purchase Orders can only be send to Suppliers that are enabled in the Supplier Management module.

A more complex interaction exists with the Cash Management module. In large enterprises, that are organized with multiple legal entities, the entities are participating in a so called Cash Pooling, with which entities are giving each other credits to balance out fluctuations between deposits and payments of each entity. This is visualized in Figure 2.4: the US daughter is expecting an incoming payment, that is scheduled to be transferred to the parent company, which will forward it to the German company, which will in turn use it to pay an invoice – no credit needs to be taken from the bank. For the Cash Management process to work optimal, deposits and payments flows not only need to be predicted reliably, but also need to be steered – payments can be re-scheduled depending on the payment terms, allowing to potentially gain a cash discount or to utilize trade credits.

Variations and Exceptions

In case an Approver is unavailable and has not set an absence delegation, any of their superiors can alternatively perform the approval. However, in any case and especially with absence delegations and overruling superiors, the Approval Engine has to ensure that Requesters cannot approve their own requests, so that always someone else is authorizing the expenditure.

As a variation to a single Good Receipt (GR) and a single invoice, it is typical to split large or long-running orders up into multiple deliveries or down payments. The GR can also be automated, for example before the beginning of each month to trigger an installment for a rent or a regular contract. In case of framework contracts, the unit price may be reduced retroactively after reaching a threshold, yielding a bonus.

The Accounts Payable (AP) team can change the payment terms or release a payment even without a received GR, to always be able to fulfill financial obligations, even if mistakes were made in earlier process steps. The AP team can also reject invoices, or request amends.

Business Requirements

Business Requirement 3.1: ERP Integration

The Procure-to-Pay process is part of the ERP system, and needs to be interconnected with several other ERP modules.

Business Requirement 3.2: General Accounting Standard Compliance

The Procure-to-Pay process is subject to accounting standards like the *International Financial Reporting Standards (IFRS)* internationally or the *Handelsgesetzbuch (HGB)* in Germany. These require, not only but especially, that all business transactions have to be booked continuously, completely, correctly, timely, and orderly, and have to be stored properly.

Business Requirement 3.3: Multiple Accounting Standards

The Procure-to-Pay process must support multiple accounting standards, for example, a German GmbH reports according to HGB to the German authorities, but according to IFRS to the international mother company.

Business Requirement 3.4: Immutability

Certain data, like a PR or PO, starts in a draft state and becomes immutable upon submission. In specific cases, changes may be possible by amends, that have to be approved and communicated explicitly.

Business Requirement 3.5: Internationalization

The application needs to handle various currencies, date & time formats, address formats, etc.

Business Requirement 3.6: Regional Variations

The Procure-to-Pay process needs to be customizable per region. Different rules exist for the Approval Engine, as per regional regulations some additional checks may need to be performed. For example, before contracting a freelancer in Germany, the previous and upcoming engagements need to be reviewed, to avoid false self-employments (“Scheinselbstständigkeit”).

2.4 Stakeholder Requirements

After collecting business process related requirements from the business owners, we proceed to collect requirements from other internal and external stakeholders. We will

focus on requirements related to the fundamental design of Enterprise Applications.

Information Security Requirements

The Information Security (InfoSec) discipline covers and consolidates requirements from internal and external stakeholders with respect to the handling of information.

In general, the *Information Security Management System (ISMS)* family of standards, ISO/IEC 27000 and following, provides guidelines for implementing information security controls. Any implementation has further to follow more specific regulations; for example, the *General Data Protection Regulation (GDPR)* regulates handling of personal data in the EU.

Aperture Science Laboratories Inc. performs cutting-edge research projects with many government agencies, which includes projects commissioned by defense departments. Information about and from these projects has to be protected specially, for example, the handling of confidential material is regulated in Germany based on the *Sicherheitsüberprüfungsgesetz (SÜG)*.

To avoid intentional or accidental data loss, information needs to be retained; for example, certain financial data needs to be retained in Germany for 10 years, but personal data must be stored as short as needed. Further, changes need to be recorded, so that earlier revisions can be restored. The granularity of changes may decrease over time, so that for example all recent changes are accessible but only explicitly marked versions are recorded for a long duration.

Based on the guidelines of ISO/IEC 27002, Aperture Science Laboratories Inc. has established the following minimal requirements related to Information Security for the implementation of any new Enterprise Application:

InfoSec Requirement 1: Identification and Authentication
: The user of an application needs to be identified and has to be authenticated. Authentication may only be possible from company-owned and compliant devices. Anonymous or pseudonymous access is only possible if explicitly allowed.

InfoSec Requirement 2: Authorization
: Access to data is only possible if the acting identity has been granted permission. Access is managed with different access levels, to separate read-only access from edit and delete access. Authorization may be granted selectively per database entry for

certain users, or may be granted widely to all entries of a database for all members of a department. Authorizations can be revoked.

InfoSec Requirement 3: Classification of Information

Files and database entries can be labeled with one of four classification levels: PUBLIC, INTERNAL, RESTRICTED, SECRET.

Only PUBLIC information can be accessed without authentication. INTERNAL information can be accessed by every employee. Access to RESTRICTED information has to be authorized. SECRET information must not leave the country of origin, and must only be handled on isolated systems, that have no access to the company-wide network or the internet.

InfoSec Requirement 4: Retention and Removal

Files and database entries can be labeled with both a minimum retention period and an automatic removal period. Information within the the retention period cannot be removed; it may be hidden and require administrative authorization to be accessed or restored. Data will automatically be removed after the given removal period.

InfoSec Requirement 5: Backup and Restore

Information is regularly backed-up, to be able to restore a known good state after an incident that involved data loss. All explicitly marked revisions are kept in every backup; unmarked and draft revisions may be kept out of backups.

IT Requirements

The implementation and operation of Enterprise Applications falls under the responsibility of the IT department; for the internal software development, the IT department of Aperture Science Laboratories Inc. has a dedicated *Development and Operations (DevOps)* team. The operation of externally developed applications is typically contracted to the vendor or a partner of them and covered with a Service-Level Agreement (SLA).

For the internal software development, the DevOps team tries to limit the number of used externally developed dependencies, as each new dependency needs to be understood by the developers, requires maintenance in case of updates, has overhead to resolve issues, and increases the risk of supply chain attacks. Further, the license of each dependency and their dependencies needs to be reviewed, as some licenses like the *GNU General Public License (GPL)* are restrictive and prohibit certain usage, or like the *Business Source License (BSL)* costly, which in case of Akka can be up to three thousand USD per CPU core

per year [1], which is unreasonable for desktop applications that are deployed to each employee.

The IT landscape of Aperture Science Laboratories Inc. is dominated by Microsoft, their Office products are used by every employee on a daily bases, and Microsoft Entra ID[1] is used for identity management and single sign-on. Microsoft Azure Cloud is used for the hosting of less critical and less sensitive applications, but own regional data centers and local servers in each office are operated for critical and sensitive applications.

IT Requirement 1: Licensing

The license of any externally developed software has to be respected. Licensing of commercial software requires approval beforehand.
Software dependencies must not use the GPL or AGPL license. Dependencies licensed with the Apache or MIT License can be used without pre-approval; software that is licensed with any other license, including LGPL and other modified GPL licenses, needs to be reviewed and approved before it is used.

IT Requirement 2: DevOps

Internally developed software has to use using *Continuous Integration (CI)*, which automates build, test, review, and deployment steps. During review, dependencies are scanned for incompatible licenses or reported vulnerabilities. Isolated environments for development, testing and production must be used.

IT Requirement 3: IT Landscape Integration

Internally developed software has to integrate well with the existing IT infrastructure. This includes the use of the company wide single sign-on mechanism, use of existing identities and group management, and the possibility to deploy the application and its updates to end devices.

IT Requirement 4: IT Infrastructure Integration

Enterprise Applications must scale for an application-specific expected number of simultaneously active users, where the load shifts globally over the duration of a day. Externally developed applications may be operated in a cloud that is managed by the vendor. If the criticality and sensitivity of an internally developed applications permits it, it may also be deployed to a cloud, else it has to be deployed to the own data centers, where it could be on bare metal or virtual machines. Hybrid deployments may be possible.

[1] previously called Microsoft Azure Active Directory

IT Requirement 5: Application Integration
Internally developed software should have interfaces to integrate with other applications. These interfaces may also be used for backup and restore operations.

2.5 Local-First Ideals

After collecting requirements for Enterprise Applications in the previous sections, we will now look at the characteristics of Local-First Software.

The notion of *Local-First Software* has been introduced in 2019 by Kleppmann et al. [11] with "seven ideals to strive for", that we will summarize here briefly without reflection. We will not yet mark these as a requirement, as we will discuss how these ideals match with the requirements of Enterprise Applications in the following Section 2.6.

Local-First Ideal 1: No spinners: your work at your fingertips
The primary copy of the data is kept locally, therefore the user does not observe a latency that would be induced by a network. Synchronization with other devices occurs transparently in the background.

Local-First Ideal 2: Your work is not trapped on one device
Users who have access to multiple local devices can switch working between them seamlessly.

Local-First Ideal 3: The network is optional
The application can be started and used without a network connection.

Local-First Ideal 4: Seamless collaboration with your colleagues
Both real-time and asynchronous collaboration is possible. Strategies to avoid or resolve conflicts are implemented. This may be based on Conflict-free Replicated Data Types (CRDTs).

Local-First Ideal 5: The Long Now
Using an application is possible even when their vendor stops supporting the product, changes the terms & conditions, or when the contract with the vendor is canceled. The data is stored in a format that can be accessed even without the original application.

Local-First Ideal 6: Security and privacy by default

Untrusted third parties, like a cloud operators, may be used to store and exchange data, by employing end-to-end encryption, so that the third party does not gain access to the content of the data.

Local-First Ideal 7: You retain ultimate ownership and control

The end user has full control over the data that is used with the application; no company controls or restricts how data is stored or processed.

2.6 Consolidation of Local-First Ideals with Enterprise Application Requirements

The seven ideals of Local-First Software have been proposed in [11] for application that are document-based or have a flat data structure, with the underlying assumptions that each document or dataset is self-contained and has a single end user as owner. This is opposite to the underlying assumptions of Enterprise Applications, that are owned by an organization and interconnected.

In this section, we will discuss the implication of the opposing underlying assumptions and transfer the seven ideals of Local-First Software to novel requirements for Local-First Enterprise Applications.

Ownership and Control

In *"Local-First Ideal 7: You retain ultimate ownership and control"*, the end user is considered as owner of the data. In case of Enterprise Applications, the enterprise takes over the role of the end user, as the enterprise ultimately owns the data. Thereby, the role of the end user shifts to act on-behalf of the enterprise, and has no ownership anymore on the data.

The idea of the seventh ideal remains however: as the owner of the data, no third party should control or restrict usage of the data. While some measures like automatic scanning and flagging of content as abusive or illegitimate is undesired for end users, it may be in the interest of an enterprise. This includes for example the automatic detection of malware in uploaded files, which may be performed by a vendor of a cloud-based Document Management System (DMS). In these cases, such measures still need to be fully controlled by the enterprise, to minimize disruptions of legitimate operations.

Local-First Enterprise Application Requirement 1: Ultimate Ownership

The enterprise has the ultimate ownership and all responsibilities. Third parties must not control or restrict how data is stored or processed without explicit assignment, in which case the control remains at the enterprise. The enterprise controls and restricts how end users can use the data.

Trusted Access

The basic motivation of *"Local-First Ideal 6: Security and privacy by default"* is applicable to Enterprise Applications, as an enterprise may rely on third parties for the operation of their applications, and data breaches or rouge employees on the side of those third parties have to be considered and are occurring in practice. The proposed mechanism of end-to-end-encryption may be a strategy to implement some aspects from *"InfoSec Requirement 2: Authorization"* and *"InfoSec Requirement 3: Classification of Information"*, especially to allow the exchange of RESTRICTED information over the network and with devices of employees that may not have permission to access the exchanged information.

The assumption of avoiding large centralized datasets contradicts however the reality of Enterprise Applications, where certain operations are performed on large datasets and need access to all entries. Encrypting different entries with different secrets just moves the problem from the primary copy to the operating instance, with an induced overhead of the encryption. In practice, the primary copy and the operating instance are typically placed in the same environment, i.e., either both are in the same data center or in the same cloud.

Encrypting different entries with different secrets also makes *"InfoSec Requirement 5: Backup and Restore"* harder or even contradicts it: in order to be able to backup and restore all information, all secrets need to be able to be backed-up and restored.

Lastly, using the available data as a source of machine learning systems is in case of Enterprise Applications not something to avoid, but a desired feature – given that the trained model is used solely for the originating enterprise and not for competitors.

Local-First Enterprise Application Requirement 2: Security

Data must be encrypted when it is exchanged with less trusted entities.

Local-First Enterprise Application Requirement 3: Full Access

Centralized trusted entities with full access may exist.

Local Data versus Centralized Data

Given that for Enterprise Applications the ultimate control does not lay at the end users, but at the enterprise. The primary copy cannot be on a local device as idealized in *"Local-First Ideal 1: No spinners: your work at your fingertips"* and *"Local-First Ideal 3: The network is optional"*, but has to be at a centralized and secured location under the control of the enterprise. This also eases the fulfillment of the requirements of *InfoSec Requirement 1* to *InfoSec Requirement 4*. Further, the amount of data of an Enterprise Application is of a different scale than in [11] considered. It is not feasible to always synchronize the full state with all devices, as end users are typically working only on a limited set of entries at a time, and untouched entries loose relevance over time.

The underlying ideas of the first and third Local-First ideals are however applicable. With data locally available as a secondary copy, the latency can be reduced, and some operations may not need to be synchronized with any other device, for example writing an Initial Incident Report. These changes can still be synchronized with other nearby devices. Other operations, for example finalizing and submitting an Enriched Incident Report, do require a coordination with other users to avoid and consolidate conflicts and therefore a synchronization with the primary copy, to ensure the report is based on the most recent state and will be recorded and communicated properly.

Local-First Enterprise Application Requirement 4: Primary Copy
The primary copy is stored at a secured centralized location.

Local-First Enterprise Application Requirement 5: Local Copies
Secondary local copies of the primary data can be created and synchronized with the primary copy. This can be restricted or limited, for example, based on the classification of the data and the authorization of the device's owner.

Local-First Enterprise Application Requirement 6: Local Operations
Operations, that do not require a synchronization with the primary copy, can be performed locally, even with a degraded network connection or with none at all.

Local-First Enterprise Application Requirement 7: Relevant Copies at Hand
Only relevant portions of the state are available locally. This state is kept up-to-date automatically. Data that is unavailable can be fetched over the network.

Local-First Enterprise Application Requirement 8: Awareness for Synchronization
The user is made aware of a pending synchronization, especially before closing the

application. The user may have a possibility to prepare for a phase of offline working, for example by fetching desired portions of the state in advance.

Roaming and Collaboration

The ideas of *"Local-First Ideal 2: Your work is not trapped on one device"* and *"Local-First Ideal 4: Seamless collaboration with your colleagues"* apply to Enterprise Applications, limitations arise however when confidential information is involved, as *"InfoSec Requirement 3: Classification of Information"* mandates that certain information is limited geographically or per device type. Similarly, collaboration has to respect the *"InfoSec Requirement 2: Authorization"*, which restricts the circle of persons to collaborate with.

An aspect to consider with collaboration is that real-time collaboration, e.g., having a shared text document and seeing the cursor position and changed text of every participant with minimal delay, is not always the desired workflow. While that can be desired to distribute the workload and to support creativity, other situations require a single user to have exclusive access and to work uninterruptedly, perhaps to restructure a longer text, to finalize a document before submission, or to avoid conflicts when working offline.

Local-First Enterprise Application Requirement 9: Nearby Copies

Secondary copies can be synchronized with each other, even without a connection to the primary copy. This can be restricted or limited, for example, based on the classification of the data and the authorization of the devices' owner.

Local-First Enterprise Application Requirement 10: Collaboration

Both real-time and asynchronous collaboration is possible. Exclusive access is possible to restrict collaboration. Collaboration respects the classification of the data and the authorization of the participants.

The Long Now

The idea of *"Local-First Ideal 5: The Long Now"* also applies to Enterprise Applications, and is further supported by *"InfoSec Requirement 4: Retention and Removal"* and *"InfoSec Requirement 5: Backup and Restore"*. This is however limited by legal requirements and *"IT Requirement 1: Licensing"* – not every externally developed software permits using it after end of the contract, even if it would be technically possible.

This can be avoided by purchasing usage rights of the externally developed software, for example by having bespoke applications developed as contract work, or by entering license agreements that permit a continued use after the end of the contract.

Local-First Enterprise Application Requirement 11: The Long Usage
: Externally developed software is purchased or licensed with indefinite and irrevocable usage rights. The application itself can be stored in a state that allows future usage.

Local-First Enterprise Application Requirement 12: The Long Storage
: The data is stored in a format that can be accessed even without the original application.

Reflection

In summary, we see in a shift from the idealized decentralized data storages with control for the end users over to centralized systems with centralized control, which is exactly what Local-First Software aimed to avoid in the first place and is similar to traditional enterprise architectures.

This raises the question, if the requirements of Enterprise Applications are compatible with the ideas Local-First Software, i.e., if the requirements developed in this section are satisfiable, and underlying if the combination can still be considered as Local-First Software. Assuming that the combination is possible, the next question that arises is if it beneficial, i.e., if the effort that has to be spend for the initial development of Enterprise Applications to support the remaining ideas of Local-First Software will yield a sufficient improvement over the traditional enterprise architectures.

Only from looking at the requirements, we cannot argue whether the proposed combination is possible or not, and need an artifact to examine this further.

3 Implementation

In the previous chapter, we have looked at three business processes of the fictional company Aperture Science Laboratories Inc. and established requirements for potential Local-First Enterprise Applications. Reflecting the identified requirements, we see two primary differences between Enterprise Applications and the original ideals of Local-First Software: Enterprise Applications have a much larger scope, with thousands of users that interact with the same shared state, and Enterprise Applications require centralized control of the data.

We presume, that Aperture Science Laboratories Inc. wants to investigate if and how Local-First Software fits into their enterprise architecture and business strategy, by investing in the development of a prototype. The prototype is called Incident Report Application and is inspired by the business process "Incident Report", which was presented in Section 2.2 on page 10. This business process was selected over the other two presented business processes, as it is the least complex process and still has representative requirements regarding the authority over data, which includes challenges about confidentiality and privacy.

The investigation and development is motivated by the central question **"Is it possible to develop Local-First Enterprise Applications?"**, which we will break down into two more concrete questions:

1. Can Local-First Software be scaled for hundreds or thousands of clients?

2. Can Enterprise Applications be developed based on the principles and ideals of Local-First Software?

3.1 Overview

A keystone in the design of this implementation is, that the Incident Report Application should continue operating with no or limited network connections, as per *"Local-First Enterprise Application Requirement 5: Local and Nearby Copies"* and *"Local-First Enterprise Application Requirement 6: Local Operations"*. When for example a local network is available, but (temporarily) no connection to the primary copy is available, it should still be possible to replicate the state within the local network and perform basic operations. Based on this, the connection and replication management is designed based on the idea of a peer-to-peer network, and only later on extended to scale out and to synchronize with the central primary copy.

The Incident Report Application is based on the REScala project [5, 6], which has an existing library for state-based Conflict-free Replicated DataTypes (CRDTs) and provides several examples for Local-First Software. Like the existing REScala examples, the Incident Report Application bases its network communication on the Communicator modules from the ScalaLoci project [8, 9], which provide direct peer-to-peer connections for Remote Procedure Calls (RPC) between so-called Registries.

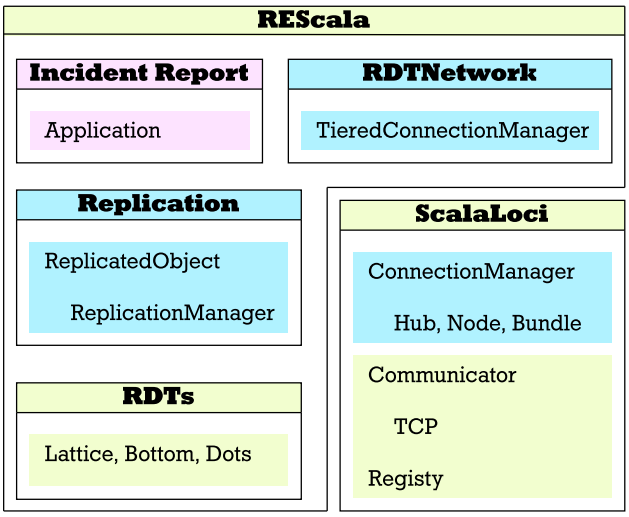


Figure 3.1: Architecture

In Figure 3.1 on the side the high-level overview of the overall architecture is shown. The Incident Report Application is highlighted in light purple; it uses all other components directly or indirectly. The previously existing components are highlighted in light green. From the ScalaLoci project, the Registry class is used to bind values and RPC-endpoints to, and serves as an anchor for incoming and outgoing connections. Connections are established using the TCP-Communicator.

Newly created libraries are highlighted in light blue. Within the ScalaLoci project, a new module for connection management has been added, which will be presented more detailed in Section 3.2 on the following page. The ConnectionManager trait and the Bundle class provide abstractions for connection management, the Hub & Node classes provide an implementation to directly connect Registries one-to-one.

Within the REScala project, a Replication module has been added, this will be presented more in detail in Section 3.3 on page 33. The ReplicatedObject trait provides a general abstraction for developing applications that use state-based CRDTs. The ReplicationManager class provides an implementation, that is based on the ConnectionManager from ScalaLoci. The module RDTNetwork will be presented in Section 3.4 on page 36. It provides extensions of the ConnectionManager, that organize the peers in a tiered tree hierarchy and dynamically exchanges the network topology and connector information with an RDT.

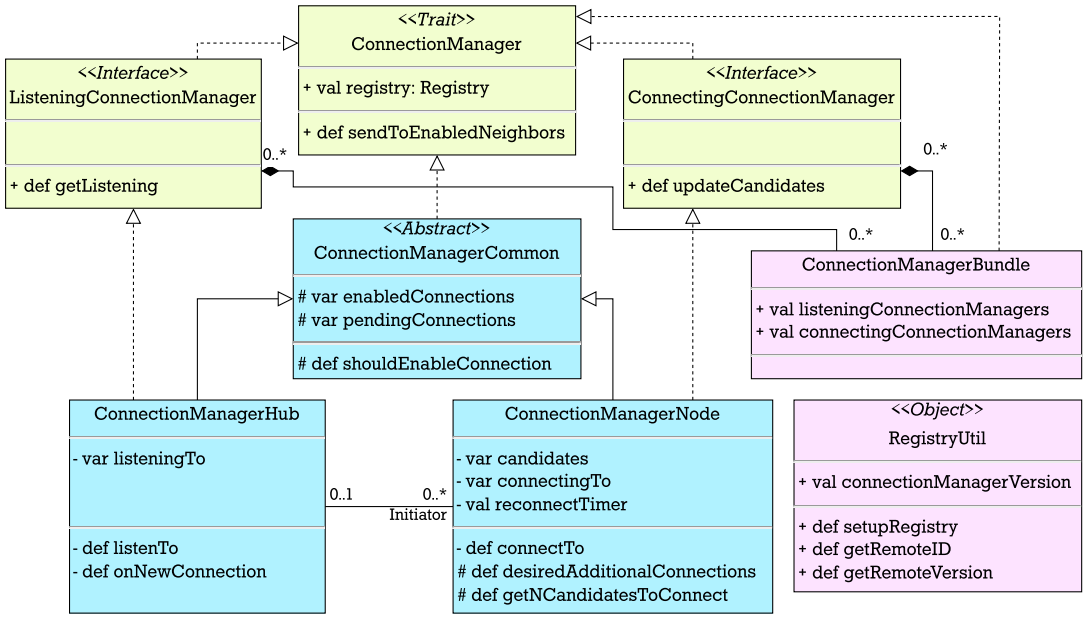


Figure 3.2: ConnectionManager Class Diagram (reduced)

3.2 Basic Connection Management

For connection management, an extensible library is added to ScalaLoci. The primary design goal of this library is to be flexible, so that it can be used in many different scenarios, ranging from simple client-server to complex peer-to-peer topologies.

As visualized in Figure 3.2, the *ConnectionManager* trait is at the root, and has a *Listening* and a *Connecting* specialization. These traits, highlighted in light green, provide a topology-agnostic interface to connect Registries and send messages between them. The method name *sendToEnabledNeighbors* includes “neighbors” to emphasize that on this low level, only single-hop delivery between directly connected Registries is supported, and “enabled” to emphasize that implementations are free two have different implementation-defined kinds of connections, of which only the “enabled” connections should be used by default to communicate on from an upper level. With this, an implementation could mark a connection as “enabled” only when certain criteria are met, for example, after a handshake between neighbors.

The *getListening* method returns a sequence of "Listening" objects, that (if the underlying communicator supports it) can be converted and serialized to a String, passed to another device, and be converted to a set of "Connector" objects, that then can be used for the *updateCandidates* method, so that this other device can establish a new connection to it. The passing of the Connector information between devices is not implemented as part of this ScalaLoci module, but will be presented later in Section 3.4 on page 36 as part of the REScala project.

Each ConnectionManager instance is tied to a single Registry, but a Registry can have multiple ConnectionManager instances, which allows for example a Registry to have several different ListeningConnectionManagers. For convenience for the implementation of more complex and dynamic topologies, each Registry gets assigned a Universally Unique Identifier (UUID), called the RegistryID. The first ConnectionManager that is being set up on a Registry can decide this ID while calling the *setupRegistry* method of the *RegistryUtil* object; in production this should be a random ID, but for testing & evaluation purposes a predefined ID may be used. This ID is stored within the Registry alongside a version field, both can be looked up remotely from a different Registry. To easier handle having multiple ConnectionManager instances on a Registry, these can be composed together with the *ConnectionManagerBundle* class, which delegates all calls and messages to all of its underlying ConnectionManagers.

A basic implementation is provided with the (abstract) classes highlighted in light blue in the Figure 3.2 on the preceding page: the abstract class *ConnectionManagerCommon* offers common methods to keep track of the state of connections, and supports enabling and terminating connections. The *shouldEnableConnection* method provides a basic equality check between the own and the remote version, to safeguard against older clients misbehaving in a network with future implementations. The check is performed asynchronously, and the method is indented to be overwritten by more complex implementations, which may want to validate more conditions before marking a connection as "enabled" – or refuse to enable it, in which case the connection will be terminated. Until a connection has been marked as "enabled", it is referenced as a "pending" connection, to keep track of those until termination, when they are automatically disconnected.

This common class is used as a base by both the *ConnectionManagerHub* and *ConnectionManagerNode* classes, which are used complementary. The hub listens on a given Listener, marks all new connections on it as "pending", and then tries to enable the connection, or disconnect it otherwise. The node however has a more complex implementation. By default, it tries to connect to all given candidates, and is periodically trying to (re-)connect to all of them, in case a connection could not be established or has been terminated. To

increase the potential use cases and allow more complex topologies, the node has a config parameter *desiredEnabledConnections*, which is controlling to how many of candidates a connection is tired to be established. The behavior of the node is further customizable, as the methods *desiredAdditionalConnections* and *getNCandidatesToConnect* can be overwritten to tailor it to the needs of more complex topologies, or to favor specific candidates over others.

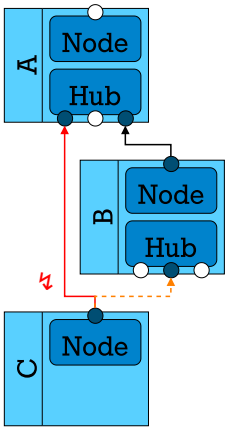


Figure 3.3: Topology Example

In Figure 3.3 on the side, a simple topology with three peers is shown as an example. The peers A and B are each using a Bundle of a single Hub and a single Node, the peer C just has a Node. All Nodes are having *desiredEnabledConnections=1*, i.e. a single connection that they want to maintain. The listeners for the Hubs and the candidates for the Nodes are assumed to be manually provided; where A has no candidates, B has A as candidate, and C has both A and B as candidates.

Initially, both B and C are having one established connection to A. As A has no candidates, its reconnection timer is not active. As B and C both have as many connections enabled as desired, they also have no reconnection timer active, even if C has another candidate it could connect to.

Once the connection between C and A fails, indicated with the lightning strike ϟ, the Node of C schedules its reconnection timer, and then Node C will pick a random candidate to connect to. In case of such a connection failure, a node does not immediately try to reconnect, to avoid overloading the peers and the network with a constant loop of establishing new connections in a case where the connection gets terminated shortly after being established.

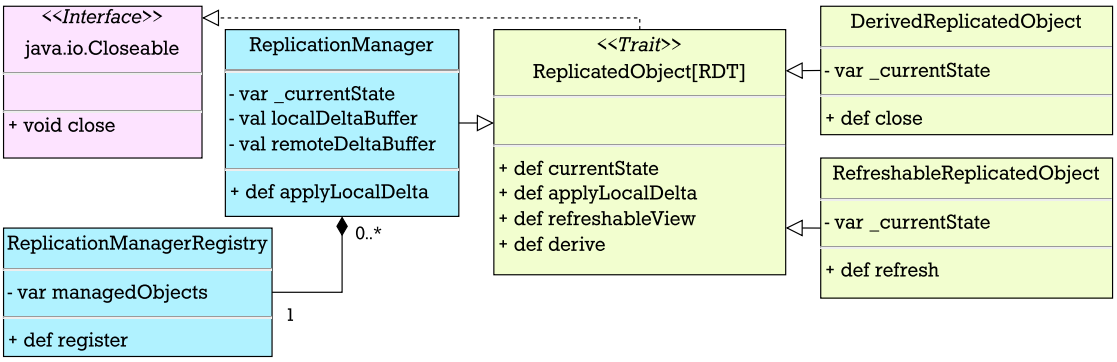


Figure 3.4: ReplicatedObject Class Diagram (reduced)

3.3 Replication Management

The state-based Conflict-free Replicated Data Types (CRDTs) of the REScala library are based on the principle, that a state is changed by mutation functions, that generate deltas, which are merged back with the state to yield the updated state. Deltas are actually of the same data type as the state, which means several deltas can be merged together to yield a possibly more compact delta, i.e., deltas can be aggregated. Both states and deltas can be serialized and exchanged over the network with other peers.

We introduce a new class *ReplicatedObject* to work with instances of an RDT, a reduced class diagram is shown in Figure 3.4. To coordinate how the locally generated deltas are send to neighbors and to merge remote deltas back to the local state, a new *ReplicationManager* class has been implemented. Each Replication Manager instance manages a single object; to have the possibility to replicate multiple (independent) objects, an *ObjectId* is used to refer to each with a well-known String. The singleton object *ReplicationManagerRegistry* has been developed to keep track of which ObjectId is managed on which Registry by which Replication Manager, and facilitates a subscribe mechanism between the peers.

Access to a Replicated Object has to be synchronized, as changes are occurring both locally from the application and by receiving remote updates. On top of that, the application may be multithreaded or use an Replicated Object in multiple different locations. Otherwise there would be race conditions, for example, if the application code would check two conditions, but in between the state would be changes by a remote update – the first condition may now be false, i.e., the state of the Replicated Object is different from what

the application logic assumes. A second challenge occurs especially with the design of Graphical User Interface (GUI) applications: often, RDTs are deeply nested, for example, the base might be a Map, which contains a class, which contains a field that should be displayed or edited. For RDTs to work, the mutation of the text field needs to be transported back to the original object, building up the delta of the text box first, then a delta of the containing class, then a delta to the Map, which then can be applied to the local state and also be replicated to other peers.

These two challenges are solved by introducing refreshable views and derivations of Replicated Objects. A refreshable view creates a copy of the current state that is not affected by updates from outside of it, i.e., when another peer changes the state, that change is not visible, as if the peer would be offline. The refreshable view however keeps a reference to its original object, and will apply any mutation that are generated from within the view to itself and the original object. A derivation is done with a pair of functions: the first function transforms a state, for example looks up one entry in a list, and the second function transforms deltas from the transformed state back to deltas of the original state, for example, wraps the change of a map entry to a delta that mutates the map. The derivation is always kept in-sync with the original state, i.e., external updates are updating the derived object and may invalidate the derivation, for example, when another peer deletes an entry.

The Replication Manager collects and applies deltas that are generated locally or received from externally. In both cases, the deltas are buffered in an ArrayBlockingQueue, so that the thread from the application / GUI and network can quickly continue with their operation and are not blocked by each others operation or by any callbacks that are fired when the Replicated Object is changed. To avoid overloading the network or the application with too many changes, the handling of buffered deltas is debounced; deltas collected during runs are aggregated to further reduce the network load and number of local callbacks. To avoid duplicated delta messages and possible broadcast storms, for example when neighbors form a ring, are all received deltas compared with the local state and only handled if they contain new information.

In Figure 3.5 three independent sequences are shown. The first sequence shows the initial state synchronization, when a second neighbor connects. The second sequence shows how a received delta from the second neighbor is received, compared with the local state, further distributed to the first neighbor, and then applied to the local state and finally applied to the UI. The third sequence shows how an action from the UI performs a local state change, that will update the UI and will be communicated to both neighbors.

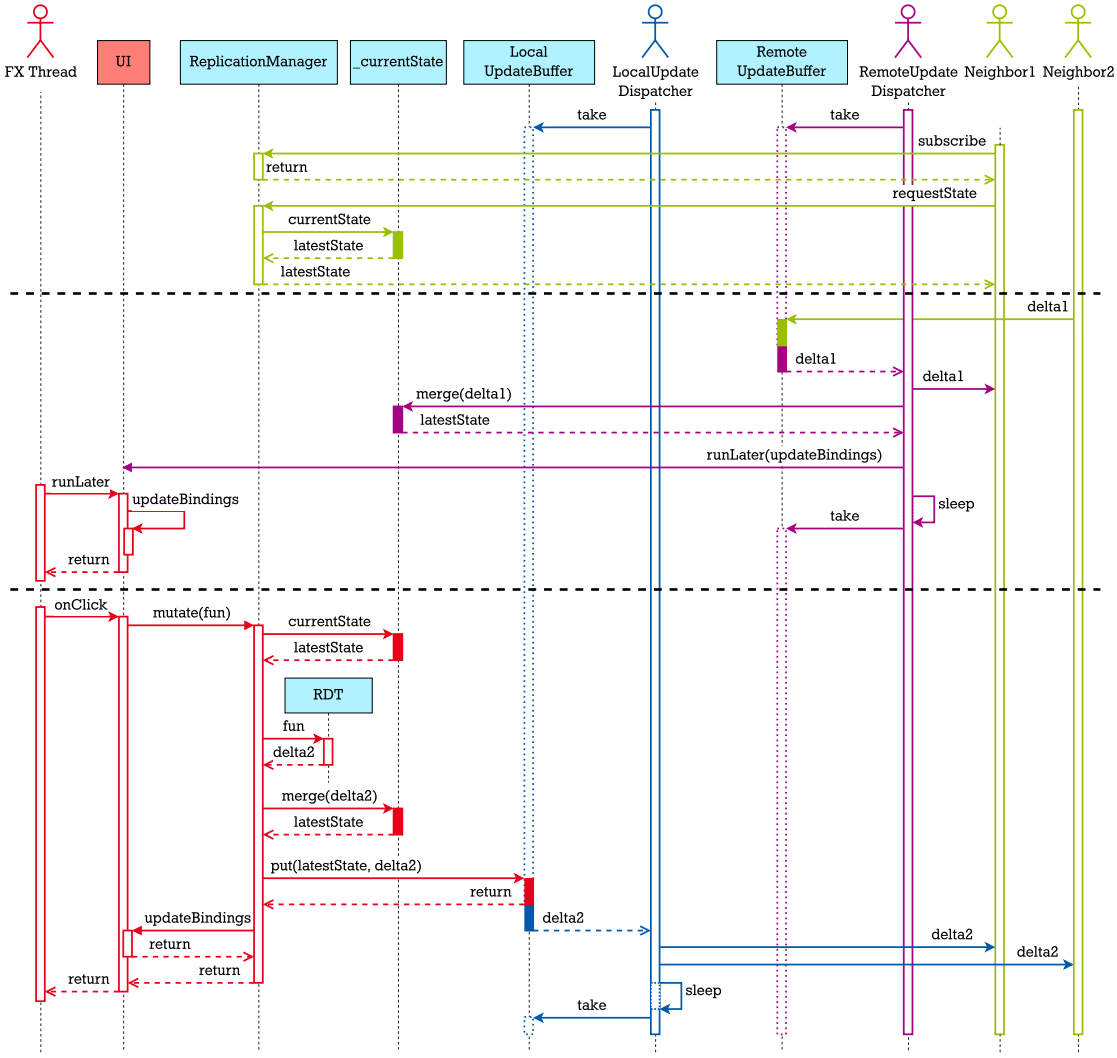


Figure 3.5: ReplicationManager Sequence Diagram

3.4 Advanced Connection Management

The basic connection management presented earlier in Section 3.2 provides the foundation to connect to other peers and exchange data with directly connected neighbors, but has no preferred topology nor mechanisms to exchange connector information. With the default setting of connecting to all known candidates, a full mesh topology would form, when all peers have every other peer as a candidate. This might be beneficial for small networks due to low latency and high resilience, but does not scale with a growing number of peers, as the number of connections grows quadratic.

Typically, Enterprise Applications are used in two scenarios: employees working in the office or employees working remotely, for example, at a client site, in the field, during travel, or from home.

In general, peer-to-peer networks are scaled by introducing an overlay network, that is based on a given topology and reduces the number of direct connections. For example, with homogeneous peers[1], a ring-based topology could be used, which needs just a few direct connections, and is easy to setup for ad-hoc networks. A ring-based topology however comes with growing size with an increased latency as the overlay network does not follow the structure of the underlying network and the availability is threatened by peers leaving the network (churn).

In case of Enterprise Applications, the peers are however not homogeneous: employees are using desktop and mobile devices, where the latter often operate on battery power due to which resources have to be used carefully. Even with the principles of Local-First Software, some application logic and especially long-term data storage is not placed on the end-user devices, but more centrally on servers.

[1] similar hardware (CPU, memory, network), similar energy constrains, similar availability, similar logical roles

We presume that Aperture Science Laboratories Inc. has decided to use a tiered tree-based topology, that is shown in Figure 3.6 on the side: a powerful server at the root (in purple) can serve as a single-point of truth. Underneath of the root, intermediary peers (in green) form a tier that distributes the load, these intermediaries can be scaled out and could for example be placed in each office, data center, and near the VPN endpoints. The end-devices of the employees are then forming the leaf-tier (in blue) and are connecting to the intermediary-tier.

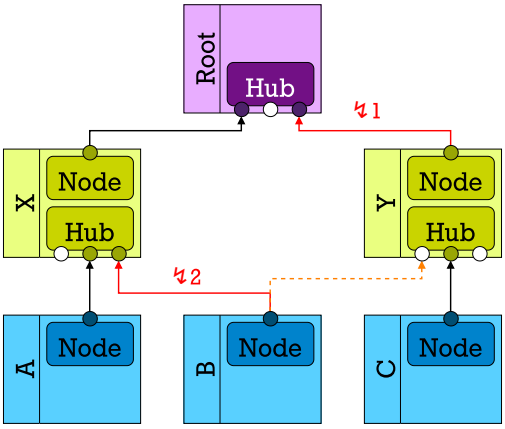


Figure 3.6: Tiered Topology

This topology has been implemented with the *TieredConnectionManager* class in the REScala project, which extends the Hub and Node classes. For the implementation two design decisions have been made, that are not ideal for use in practice, but benefit the use in a demo / prototype setting: 1. peers can only connect connect to peers located on a directly upper tier, i.e., A can connect to X but not to Root, 2. peers desire & enable only 1 connection, and 3. peers have no preference to which of the possible other peers they connect to, i.e., B can connect to either X or Y (but not to both at the same time). The implementation has no preference for the number of tiers. For evaluation and for practical use the three design decisions can be overwritten, for example, it would make sense to organize peers geographically and have an intermediary that is located in an office only be available for clients within that office.

When the connection from the peer Y to the Root fails, as indicated with the first lightning strike ϟ1, then will the right branch form a partition that is not connected to the rest of the network, as peers can only connect to upper tiers and so Y cannot connect to X. When the connection from B to X fails, as indicated with the second lightning strike ϟ2, then will B attempt to connect to Y.

For this to work, each peer needs to know the connector information of the peers on their upper tier. This is solved by storing the connector information of each peer in an RDT and replicating it to all peers and implemented with the *TieredReplicatedConnectionManager* class. To ease the bootstrapping, intermediaries can seed their current RDT state to any peer, regardless of their tier. Each peer adds its own connector information, alongside

some more fields for debugging, to the RDT. Whenever a peer receives updates for the RDT, it refreshes its available candidates.

3.5 Incident Report Application

Looking back at the original motivating question “Is it possible to develop Local-First Enterprise Applications?”, we now have the underlying libraries for connection management and replication management, so that we can build on those and develop an application for the further evaluation of that question. We are not interested in a fully functional Enterprise Application – to evaluate and discuss how the requirements of Enterprise Applications fit with the ideals of Local-First Software, we only need a representation of the basic principles.

The Incident Report Application has been developed as such a prototype, targeted for the internal DevOps team of Aperture Science Laboratories Inc. It is inspired by the business process “Incident Report”, which was presented in Section 2.2 on page 10. The current implementation is focused on the evaluation of the connectivity and replication, and to serve as a discussion artifact for the remaining requirements.

The Incident Report Application is developed in Scala and uses JavaFX for the Graphical User Interface (GUI). Two screenshots are taken at the same time, the Figure 3.7 on page 40 shows the state of the peer AAAA and Figure 3.8 of the peer BBBB. The differences in the displayed date & time fields comes from applied internationalization: the peer AAAA has been started with the local German timezone, the peer BBBB has been started with setting the “user.timezone” property of the JVM to “Asia/Tokyo”. The main window of the GUI is shown behind, it is organized in three columns: on the left side is a general menu, from which a new blank incident report can be created. In the center is a table of all incident reports, the table can be sorted and one entry can be selected. The selected entry is shown on the right side in a read-only live state.

With the “Edit” buttons on the right panel, an edit dialog can be opened, which is visible on top of the main window in each screenshot. As REScala offers no specialized datatype for text fields, *LastWriterWins[String]* has been chosen for text fields. This is not optimal, as conflicting changes overwrite each other and do not preserve any history. The difference between the several “Edit” buttons will be discussed in the following Section 3.6, which will also explain why the field “Investigation” in the edit window of the peer BBBB is empty.

3.6 Asynchronous Application Design

The Incident Report Application as a prototype of a Local-First Enterprise Application gives a framework to discuss how some of the remaining requirements could be implemented.

As discussed in Section 2.6 with the *Local-First Enterprise Application Requirements 4* to *8*, certain actions in Enterprise Applications require an explicit synchronization with a central entity. Looking at the the Incident Report Application, the confirmation of the severity of an Initial Incident Report by the Line Manager is such an action, as this confirmation can be done only once, has to be based on the latest revision of the report, needs to be recorded and retained, and is the trigger for other actions. If the confirmation were to be send without any central synchronization, it could be factually wrong as the confirmed severity might not match the latest description, there could be multiple conflicting confirmations, or the confirmation could get lost due to network issues.

To match the ideas of Local-First Software, it should however be possible perform actions asynchronously, for example, to confirm the severity even when the device has no or an unreliable network connection, and have this confirmation processed at a later time. Processing actions asynchronously raises two challenges: first, the user must be kept informed about the progress of their action, and second, an action might become no longer desired after some time, or become invalid due to conflicting other changes.

Related to this, collaboration can be real-time and asynchronous, as outlined with *"Local-First Enterprise Application Requirement 10: Collaboration"*. Conflicting changes could be avoided with exclusive access for a single user.

We have started to investigate possible implementations to address the requirements for asynchronous work, but not have come to a complete solution and leave this to be picked-up by a future work.

In a first step, we propose that the User Interface (UI) should allow the user to choose a desired collaboration mechanism. The Figure 3.7 on the following page shows on the right side for a selected entry three options for editing:

Edit Live opens the edit modal with a bidirectional connection. If this is done by multiple users that are connected, the change of any user will be visible for the other users, and someone else can continue editing based on the change. We propose this mode for real-time collaborative work, but the current implementation is limited by the

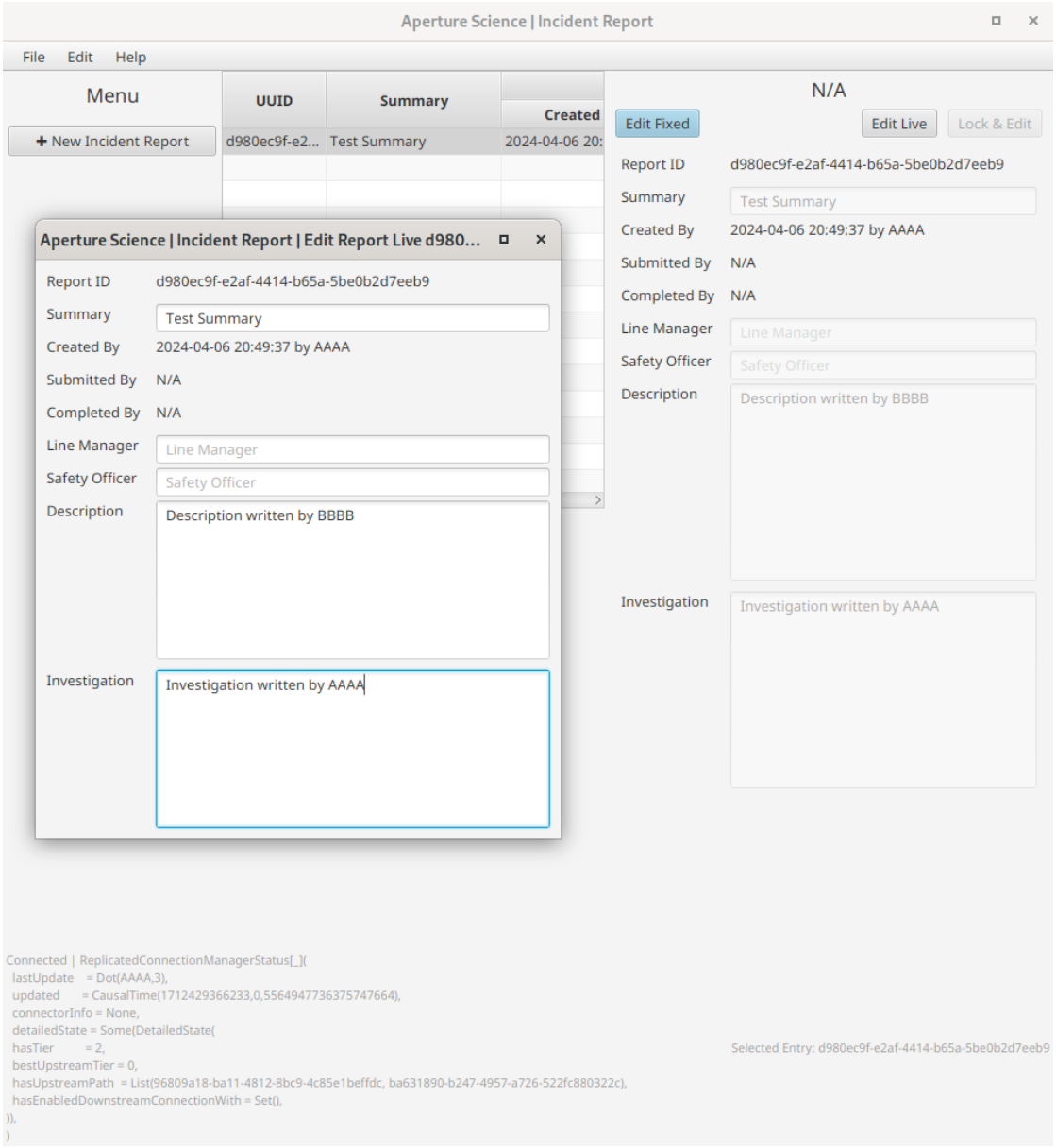


Figure 3.7: Screenshot of the INCIDENT REPORT APPLICATION – Peer AAAA

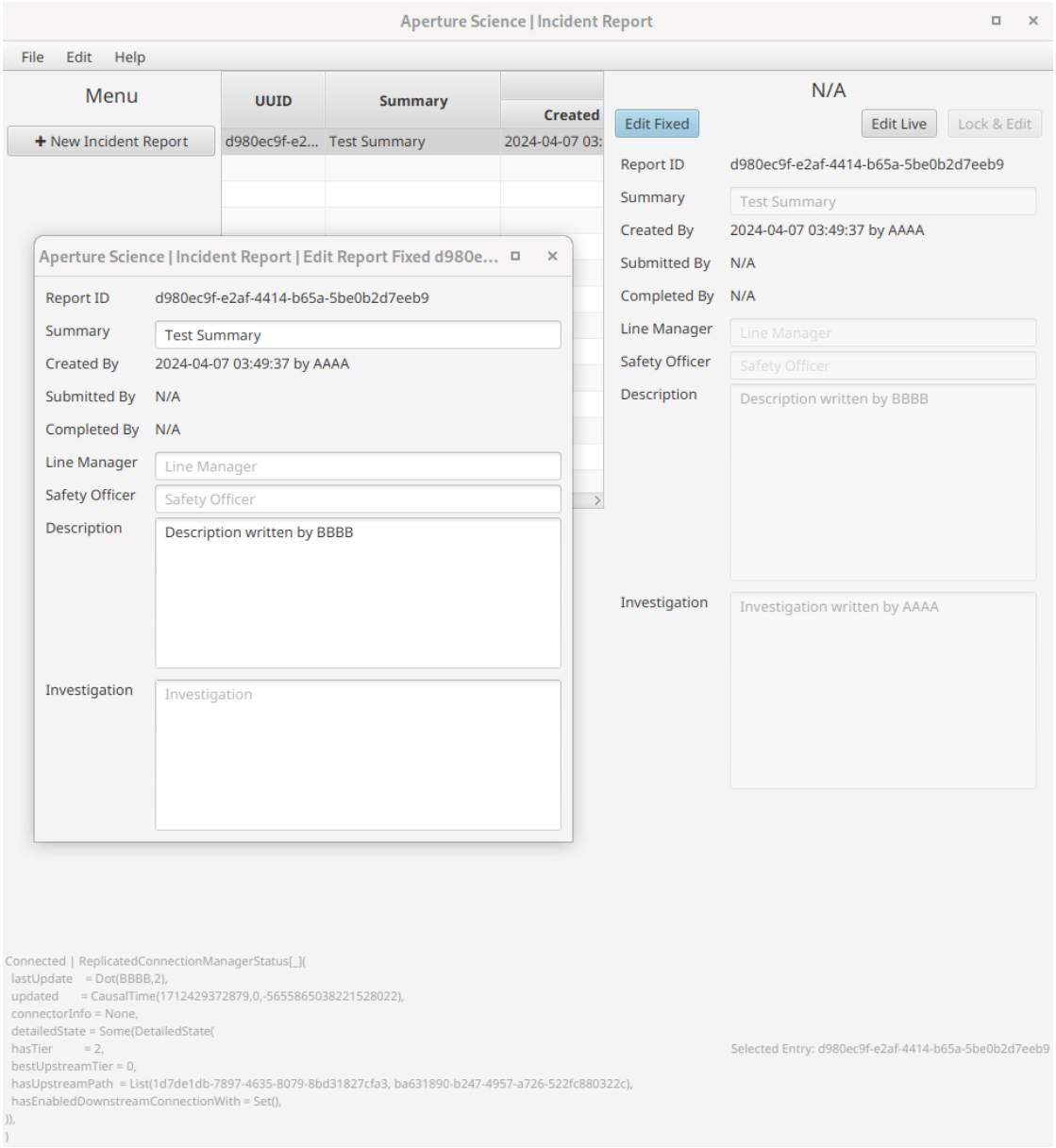


Figure 3.8: Screenshot of the INCIDENT REPORT APPLICATION – Peer BBBB

chosen underlying LastWriterWins data type, which simply resolves conflicting changes to the latest one.

Edit Fixed opens the edit modal with a unidirectional connection. Externally created changes will not arrive at the edit modal, i.e., the user sees only their own changes. Changes created in this modal are however propagated to other users, so that they can see the progress. We propose this mode as first option for uninterrupted work, but again the current implementation is limited by the chosen underlying LastWriterWins data type, with which the problem of conflicting changes is even amplified, as other user's changes are never displayed.

Lock & Edit is not implemented. It is supposed as a second option for uninterrupted work, by requesting exclusive access for an entry.

In the aforementioned two screenshots, both users opened the edit modal at the same time. On Peer AAAA it was opened in the live mode, on peer BBBB in the fixed mode. Changes from peer BBBB are visible in the window of peer AAAA. The addition of "Investigation written by AAAA" from peer AAAA arrived on peer BBBB, as it is visible in the main window, but is not presented in the edit modal.

As a second step to address asynchronous work, we looked at potential mechanisms to submit actions asynchronously and to request exclusive access. We encountered challenges related to exclusive access and have not integrated this work into the Incident Report Application, but have already developed some test cases, to be further developed by a future work.

4 Evaluation

In the previous chapter, we have looked at the implementation of the Incident Report Application and its underlying libraries, which we have developed to bring us closer to an answer for the motivating question *“Is it possible to develop Local-First Enterprise Applications?”*, that we broke down into the two more detailed questions *“Can Local-First Software be scaled for hundreds or thousands of clients?”* and *“Can Enterprise Applications be developed based on the principles and ideals of Local-First Software?”*.

Even in the broken down form, these questions are still too big to answer positively with the current state of the implementation, but we can also not rule out the opposite. In this chapter, we will look closer at some parts of the implementation and look at some important remaining challenges.

4.1 Connection Management and Performance Benchmark

To investigate if the connection management and replication management implementations are suited as base for large-scale deployment, we have developed a benchmark application.

During the implementation of the ReplicationManager, we assumed that a debounce period could beneficial for the overall performance, and had to estimate values for the period. In a first step, we are looking at how to tune the values for the debounce period, and to get a general feeling of the unoptimized performance.

The benchmark is constantly creating new Incident Reports on one instance, and on a second instance every new Incident Report is marked as “completed”. The first instance measures the delay between the creation and the “completed” mark. The second instance is measuring the number of marked reports. Both instances are connected over the Root intermediate. All three replicas have the same debounce values. In the benchmark, we

vary the debounce period and the rate by which the first instance creates new reports, by having a delay between each creation. The benchmark is performed with all three instances on a single device. the benchmark is executed for 10 seconds as a warmup without taking measurements, followed by 10 seconds in which the measurements are taken. The benchmark is performed first without a dedeuplication check and then with a deduplication check that is based on “lattice.lteq”.

Without any overhead, the number of requests would be $\frac{1}{pause}$, i.e., 10 requests per second when waiting 100ms between each creation, and 1.000 requests per second when waiting 1ms between each creation.

We can see the results of the benchmark in Figure 4.1. The number of requests that are created with a delay of 100ms and 10ms match for all three debounce values the expected value of 10 resp. 100 requests. With a pause of just 1ms between the requests we however see a significant difference between the debounce values; an optimum of 1,000 would be expected without any overhead. In this case, the debounce period of 500ms with an average of 774.00 requests per second is by far outperforming the debounce period of 50ms which achieves 298.00 requests per second and the debounce period of 5ms which achieves only 147.44 requests per second. We see in all three cases, that the measured delay significantly increases in the case of waiting 1ms between the requests.

The test case of debounce period of 5ms with a pause time of 1ms was unable to complete due to serialization errors with duplication check. We assume, that the added computation with “lattice.lteq” for every delta leads to more requests being buffered during that time, which may lead to the message getting too large for the serialization. We therefore supplemented this with a pause time of 2ms.

From these results we can see, that a larger debounce period can handle more requests, at the cost of a higher delay.

During the execution of the benchmark, we have had the impression that the benchmark is limited by single CPU core performance. A more realistic benchmark with more devices is needed to further investigate how many requests can be handled reliably, and if the load can be scaled on the root node for several thousand active users at a time.

In more realistic settings, the peers would not necessarily be connected to the same intermediary. The chosen topology of a tiered tree leads to a potentially high latency between leafs as every intermediate adds not only the delay of the network but also for the debounce period. This can accumulates especially when the changes have to replicate through the root – which is also a single point of failure, threatening availability.

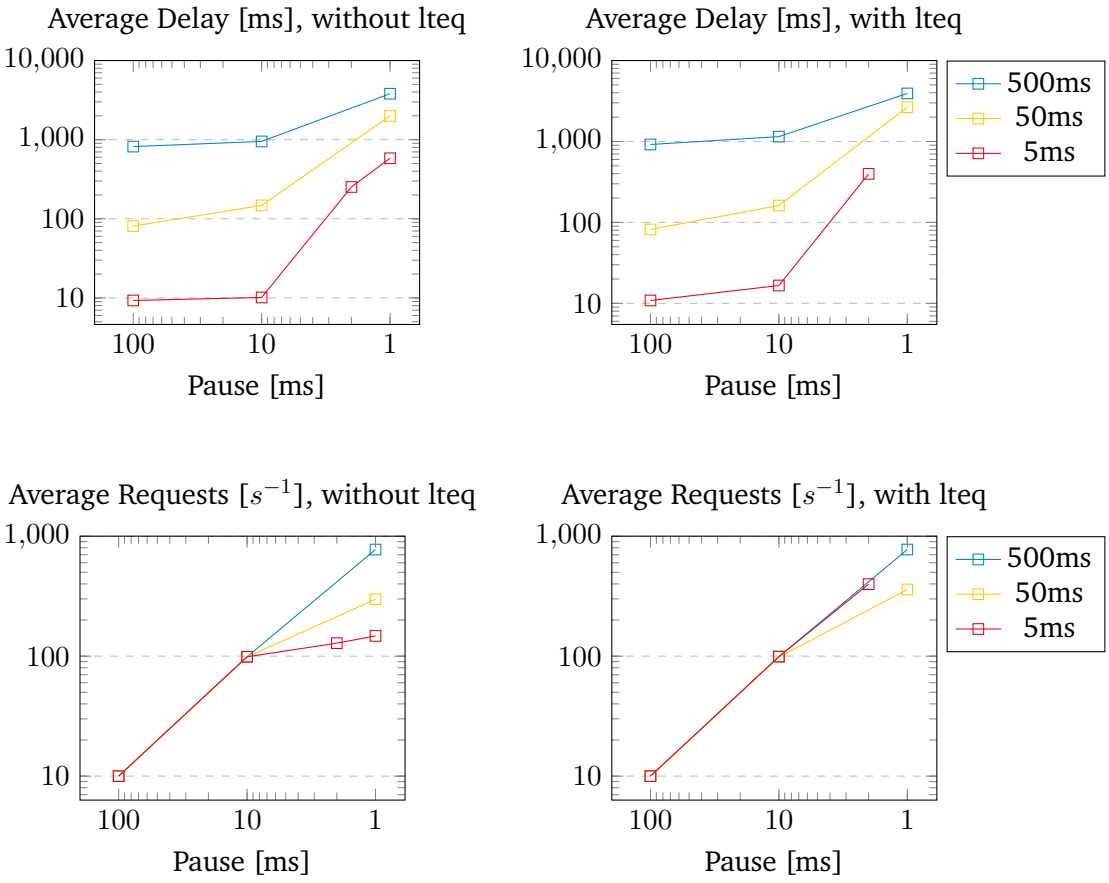


Figure 4.1: Replication Performance Evaluation

The tiered tree could be enhanced with connections between peers. For example, all members of a tier could additionally form a ring; this could be beneficial on the topmost tier below the root, especially when this tier is spread out globally and is used to interconnect continents, to potentially reduce load and increase resilience. Direct connections could also be established between the leafs: when two users are editing the same incident report, a direct connection between their devices would be a shortcut to avoid the latency incurred by intermediates.

The introduction of topologies with direct connections is already supported by the ReplicationManager, as it avoids the replication of duplicate deltas by comparing every received delta with "lattice.lteq" with the current state, so that broadcast storms are avoided.

We further performed manual tests, by running the application on a local network and over the internet. In these tests, we noticed that the (re-)connection logic of *ConnectionManagerNode* is sufficient, but not optimal. It leaves times where no enabled connection is available for longer than necessary, for example, when a connection fails a reconnection is scheduled instead of immediately trying to establish a new connection. To avoid overloading peers and the network, and to avoid connecting to a reoccurring failing hub, candidates should be blacklisted for a certain time. The downtime can further be reduced, by establishing "standby" connections, that are being kept alive, but not marked as "enabled".

4.2 Prototype Gap

Looking at the gap between the Incident Report Application as a simplified prototype with requirements for realistic Enterprise Applications, we see the biggest open challenge in the design of the underlying data structure.

The data structure of the Incident Report Application is based on an RDT of a single GrowOnlyMap. In that form, it is a standalone dataset, that is not integrated with other Enterprise Applications. Realistically, and especially to support *"Business Requirement 2.8: ERP Integration"*, an interconnected data structure would be needed. This raises questions on how this integrates and scales with many different Enterprise Applications, and with how the interdependencies can be kept in a consistent state.

To address the scale of Enterprise Applications, even in the current form without a interconnected data structure, changes to the ReplicationManager and underlying RDTs

are needed, to allow for partial replication and lazy-fetching, as it has been described with *"Local-First Enterprise Application Requirement 7: Relevant Copies at Hand"*.

Other important requirements that have not been addressed by the Incident Report Application are less related to Enterprise Applications, but more to Local-First Software in general and REScala in particular.

For collaboration, a specialized RDT for text fields is needed that decomposes and merges the individual changes, alongside an editor in the User Interface (UI) that may be used also to resolve conflicts. Further, the UI has to be extended for *"Local-First Enterprise Application Requirement 8: Awareness for Synchronization"* to indicate the replication status to the user, possibly by highlighting entries in the list if it is not known that the local entry matches the state of the primary copy.

5 Summary and Conclusion

In this master thesis, we have investigated how the ideals of Local-First Software match with the Requirements of Enterprise Applications.

In Chapter 2 we have first looked at the business requirements of three business processes, followed by analyzing more general requirements for the development and implementation of Enterprise Applications. We then discussed in detail how the seven ideals of Local-First Software from [11], which are based on the perspective of individual end users and standalone documents and repositories, apply to the context of Enterprise Applications, which have to serve the needs of an organization with several thousand employees. During this discussion, we presented a total of 12 more specific requirements for Local-First Enterprise Applications, which centralize the primary data storage and control, while using the ideas of Local-First Software to still reduce the dependency from centralized systems compared to traditional enterprise architectures.

In Chapter 3 we gradually build up a prototype, to further examine the feasibility of Local-First Enterprise Applications. We started by implementing a ConnectionManager for the ScalaLoci project, which automatically establishes and maintains peer-to-peer connections between neighbors. On top of that, we implemented a peer-to-peer topology based on a tiered tree, to scale this network out. We then implemented a ReplicationManager, that aggregates and distributes changes of an RDT with its neighbors in a peer-to-peer network, which will then propagate throughout the peer-to-peer network. In the evaluation, we have performed a benchmark to analyze the tradeoff between throughput and latency for the replication.

We proceeded to develop a prototype of an Local-First Enterprise Application, that is inspired by the Incident Report business process, to see how such an application would look and behave, and to have an artifact to further examine and discuss. The implementation is limited by missing features to support asynchronous work or real-time collaboration, and cannot scale sufficiently as the full state is replicated with every replica. Many requirements

have not yet been considered with the implementation, notably requirements related to Information Security (InfoSec).

Although the InfoSec Requirements are a must-have for real-world Enterprise Applications, the general idea of encrypting the RDT or parts of it is not specific to Enterprise Applications and applies in general to Local-First Software. Looking specifically at Local-First Enterprise Applications, we see the biggest open challenges in more complex and interconnected datastructures, and with how different Enterprise Applications, Local-First and traditional, can be integrated with each other.

In summary, we cannot conclude whether future Enterprise Applications can be based, or even should be based, on the ideals of Local-First Software or not. The combination seems however realistic and should be further investigated.

List of Figures

Bibliography

[1] Lightbend, Akka Production Pricing, . URL https://www.lightbend.com/akka/pricing. Last accessed on: 2024-04-17.

[2] European Commission: SME definition, . URL https://single-market-economy.ec.europa.eu/smes/sme-definition_en. Last accessed on: 2023-11-28.

[3] GDPR, Article 9, . URL https://eur-lex.europa.eu/legal-content/EN/TXT/HTML/?uri=CELEX:32016R0679&from=EN#d1e2051-1-1. Last accessed on: 2024-04-17.

[4] OSHA – 29CFR 1904 Recording and Reporting Occupational Injuries and Illnesses, . URL https://www.ecfr.gov/current/title-29/subtitle-B/chapter-XVII/part-1904. Last accessed on: 2024-04-17.

[5] REScala, . URL https://www.rescala-lang.com/. Last accessed on: 2024-04-17.

[6] GitHub: REScala, . URL https://github.com/rescala-lang/REScala. Last accessed on: 2024-04-17.

[7] § 193 Sozialgesetzbuch - SGB VII, . URL https://www.gesetze-im-internet.de/sgb_7/__193.html. Last accessed on: 2024-04-17.

[8] ScalaLoci, . URL https://scala-loci.github.io/. Last accessed on: 2024-04-17.

[9] GitHub: ScalaLoci, . URL https://github.com/scala-loci/scala-loci/. Last accessed on: 2024-04-17.

[10] Albert Fleischmann, Stefan Oppl, Werner Schmidt, and Christian Stary. *Ganzheitliche Digitalisierung von Prozessen*. Springer, 2018. ISBN 978-3-658-22647-3.

[11] Martin Kleppmann, Adam Wiggins, Peter van Hardenberg, and Mark McGranaghan. Local-first software: you own your data, in spite of the cloud. In *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, Onward! 2019, pages 154–178, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369954. doi: 10.1145/3359591.3359737. URL https://doi.org/10.1145/3359591.3359737.

[12] Dieter Masak. *Moderne Enterprise-Architekturen*. Xpert.press. Berlin, 2005. ISBN 3540229469. URL http://scans.hebis.de/HEBCGI/show.pl?12716853_kap-1.pdf.

[13] Hermann J. Schmelzer and Wolfgang Sesselmann. *Geschäftsprozessmanagement in der Praxis: Kunden zufriedenstellen, Produktivität steigern, Wert erhöhen*. Hanser, München, 8., überarbeitete und erweiterte Auflage edition, 2013. ISBN 9783446434608. URL http://d-nb.info/1028829019/04.

[14] Uwe H. Schneider. Das Recht der Konzernfinanzierung. *Zeitschrift für Unternehmens- und Gesellschaftsrecht*, 13(3):497–537, 1984. doi: doi:10.1515/zgre.1984.13.3.497. URL https://doi.org/10.1515/zgre.1984.13.3.497.

[15] André Wolski. Spezifikation einer Ausführungssemantik für das Subjektorientierte Prozessmanagement mit CoreASM. Bachelor's thesis, TU Darmstadt, Darmstadt, October 2019. URL http://tuprints.ulb.tu-darmstadt.de/8360/.

[16] André Wolski, Stephan Borgert, and Lutz Heuser. An extended Subject-Oriented Business Process Management Execution Semantics. In Stefanie Betz, Matthes Elstermann, and Matthias Lederer, editors, *S-BPM ONE 2019, 11th International Conference on Subject Oriented Business Process Management*, ICPC published by ACM Digital Library, pages 69–81. Association of Computing Machinery (ACM), June 2019.

[17] André Wolski, Stephan Borgert, and Lutz Heuser. A CoreASM based Reference Implementation for Subject Oriented Business Process Management Execution Semantics. In Stefanie Betz, Matthes Elstermann, and Matthias Lederer, editors, *S-BPM ONE 2019, 11th International Conference on Subject Oriented Business Process Management*, ICPC published by ACM Digital Library, pages 83–97. Association of Computing Machinery (ACM), June 2019.