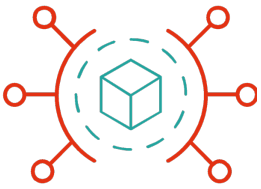

New ICT infrastructure and reference architecture to support Operations in future PI Logistics NETworks

D2.10 Blockchain Transactional Ledgers and Smart Contracts as PI Enablers

Document Summary Information

Grant Agreement No	769119	**Acronym**	ICONET
Full Title	New **IC**T infrastructure and reference architecture to support **O**perations in future PI Logistics **NET**works		
Start Date	01/09/2018	**Duration**	30 months
Project URL	https://www.iconetproject.eu/		
Deliverable	D2.10 Blockchain Transactional Ledgers and Smart Contracts as PI Enablers v2		
Work Package	WP2		
Contractual due date	31/01/2020	**Actual submission date**	-
Nature	Other	**Dissemination Level**	PU
Lead Beneficiary	ILS Group		
Responsible Author	Alexander Papageorgiou (ILS Group)		
Contributions from	CNIT, IBM, NGS, VLTN		

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Grant Agreement No 769119.

Disclaimer

Copyright message

Table of Contents

List of Figures

List of Tables

Glossary of terms and abbreviations used

Abbreviation / Term	Description
API	Application Programming Interface – A method via which computer code is able to interface with a program, usually defined for web services and system utilities
DX	Deliverable X – The ICONET Deliverable identified by the number X
ES6	ECMAScript 6 – A scripting language specification whose most widely known implementation is JavaScript
GA	Grant Agreement – The Grant Agreement of the ICONET project outlining the scope of work of the project
GPS	Global Positioning System – A system that uses satellites to pinpoint the location of an object using longitude and latitude coordinates
HF	Hyperledger Fabric – The enterprise grade blockchain implementation of the Hyperledger umbrella project, focusing on business networks and logic
JSON	JavaScript Object Notation – A specification as to how to define JavaScript objects in a standardized format that is easily converted from and to a textual format
KPI	Key Performance Indicator – A usually numerical indicator of a system's performance that is associated with an explanation of what it indicates
LL	Living Lab – A testbed where the ICONET solution is meant to be tested on
NLP	Neural Language Processing – A methodology via which natural language is analysed and contextual information is extracted from it
N/A	Non-Applicable – A case where the corresponding column of a row in a table does not have a value associated with it as it is not possible to have one
OLI	Open Logistics Interconnection – A framework that specifies how the overall logistics operators should be categorized into groups and interact with each other in a layered format
PI	Physical Internet – The "internet" as formed by the various operators and enablers of the global logistics network w/ the assistance of state-of-the-art technology and IoT
QX	Quartile X – Usually denotes the quartile of a year where something is expected to occur with X indicating the number of the quartile
RESTful	Representational State Transfer (ful) – A set of definitions that dictate how a web API is supposed to be interfaced with by providing verbose HTTP types, such as PATCH for updates or POST for creations
SHA	Secure Hashing Algorithm – An algorithm that is widely accepted to be the most secure hashing algorithm to date that produces a deterministic uniformly random output of a specific length with any type of input sequence of bytes
SLA	Service Level Agreement – An agreement that dictates certain KPIs a service is meant to achieve, usually utilized as a guarantee that a purchased service will meet a set of requirements
STX	SubTask X – The sub-task number X of a specific task as defined in a WP

TDD	Test Driven Development – A programming method of operation that requires any functionality that is coded for a program to be accompanied by tests that confirm that functionality via meaningful language e.g. “the application can do XYZ”
URI	Uniform Resource Identifier – An identifier of a unique resource within a system, usually utilized as a web URL component
URL	Uniform Resource Locator – A string that represents the location of a resource within a system, usually utilized as a website link to inform the site of which resource one wishes to access
VSM	Value Stream Mapping – See D3.6 representing LL2
vX	Version X – A numerical versioning system with X specifying the sequential version of an asset
WP	Work Package – A document detailing the work that needs to be carried out by each partner of a project

1 Executive Summary

The contents of this report consume the outputs of version 1 of the deliverable by expanding upon its findings and presenting a methodology via which it is possible to enhance the applicable value of the ICONET blockchain by associating Ricardian contracts with the ICONET blockchain's functions thus legally protecting and enforcing them.

Initially, the outputs of v1 of the deliverable are laid out and detailed as to how they will be consumed within this report. Specifically, the blockchain solution that will be used by ICONET is named, the Tendermint Consensus engine, along with the smart contract implementation of a dynamic speed limit for a packet flowing in the PI network and the corresponding LL use cases. These inputs are all justified under the pretense of Ricardian contracts to properly justify why Ricardian contracts were chosen as a means of empowering the functionality of the blockchain with legal affectability.

After the precedence is explained appropriately, the concept of Ricardian contracts is explained as well as their already-proven usability and smart contract correlation. Methodologies via which smart contracts can be linked to Ricardian contracts and vice versa are explored with the concept of PI in mind. These methodologies are then applied to each LL use case to highlight why and how Ricardian contracts are useful for the LLs with direct feedback and validation from the LL hosts.

Once the theoretical groundwork is complete, the practical programming steps taken are analysed and expanded in-depth. First, the legal enforce-ability of dynamically generated Ricardian contracts is conducted to conclude which types of Ricardian contracts ICONET will use. The candidate legal contract, after correspondence with the LL operators, is then identified and transformed to the prose necessary by smart contracts. The implementation details surrounding the dynamic generation of these contracts based on parameterization of certain variables are subsequently written.

Finally, workflows are described that map to each LL use case and indicate the resulting impacts of utilizing Ricardian contracts in existing scenarios. These workflows are tailored to the specifications of each LL and have been created by acquiring feedback from each LL participant. Code that showcases these workflows and tests them while measuring key performance indicators is also provided to better gauge the impact of the proposed solution with tangible numbers as well as logical conclusions for immeasurable-mathematically indicators such as level of security.

The version 1 of this report focused on the research activities surrounding the blockchain aspect of ICONET, however it did not contain any tangible work that relates to the actual tracking of SLAs within the blockchain context per the specifications of ICONET. This document includes this work as well as any supplementary information that was deemed necessary to include for the report to be considered complete. Version 3 will expand upon these principles to develop and test an automatic arbitration service that will utilize the codebase of Version 2.

The related effort for the above is outcome of the ICONET WP2, Task 2.4 "Blockchain mechanisms for secure and privacy-preserving distributed transactional ledgers", the subtasks concerning the realization of smart contracts & the blockchain specifications, and the related versioned deliverables 2.9 (v1), 2.10 (v2) and 2.11 (v3).

2 Introduction

The focus of this deliverable is to dictate how Ricardian contracts can be combined with the produced blockchain and smart contract solution of the priori version of this deliverable while simultaneously curating the resulting workflows to the needs of each unique LL and keeping the resulting solution as modular and widely-applicable as possible. This deliverable serves as the outcome of WP2 and particularly task 2.4 (Blockchain mechanisms for secure and privacy-preserving distributed transactional ledgers) led by the INLECOM Group (ILS Group). In this WP, as the report indicates, the versatility of Ricardian contracts as a key differentiator for the usability of the ICONET blockchain is investigated under the scopes of all LL use cases, with tangible and non-tangible outputs demonstrating the investigation outcomes.

The fundamental theoretical principles that have been devised for the deliberation process of this deliverable retain maximum modularity in mind to ensure that the deliverable's outputs are concurrently domain-specific for ICONET and usable under other non-PI contexts as achieved by enabling the Ricardian contracts that are linked to the smart contracts to either be dynamically or manually generated, bestowing greater freedom to the ICONET blockchain's operators and users. This approach is exhibited in the way the code of this deliverable operates.

This deliverable at its core tries to formalize the following key statements:

- The Ricardian contracts relevant to PI
- The legality and methodology of dynamically generated Ricardian contracts with contextual parameters
- The practicality and pros & cons of utilizing these contracts in LL scenarios

Any statements regarding the LLs have been formed after extensive discussions between the LL operators and the deliverable authors to certify that they represent the actual needs of the current market state rather than broad research.

2.1 Deliverable Overview and Report Structure

In accordance with the ICONET workplan, this report describes task 2.4 and all subtasks that pertain to it with impartiality and transparency in mind.

The deliverable's foremost objective is to consume the outputs of version 1 and consolidate and use those outputs with Ricardian contracts to create a new unified workflow that combines both technologies. This will be achieved by studying how Ricardian contracts can be applied in the context of blockchain technology and in the context of PI in general whilst curating the findings for the purposes of ICONET's LLs. The primary tangible outputs of this report are a software prototype that will initiate the ICONET blockchain developed in version 1 of the deliverable with the appropriate parameters and adaptations to support linkage with Ricardian contracts as well as a software prototype that will generate a Ricardian contract and link it to a smart contract via the API developed in version 1 and expanded upon in version 2 of this deliverable. The tool that generates Ricardian contracts will function by processing one of the following inputs:

- A Ricardian contract template with parameterization
- A Ricardian contract with dynamic terms
- A Ricardian contact created by the user

This tool will then link the contract with a smart contract identifier provided by the operator of the tool. Surplus outputs of this report include a parsing pseudo-language for creating Ricardian contract templates, a PI Ricardian contract template and full PI workflows for Ricardian contracts integrated to the LL scenarios.

At the beginning, the academic materials that were exhausted to evaluate the viability of Ricardian contracts in a PI environment are illustrated in a summarized format. Ricardian contracts have been devised since 1998, however they have seen little to no use in legal contexts as the market was not mature enough back then to fully exploit them. With the emergence of blockchain technology, Ricardian contracts have become as prevalent as

ever due to their versatility in strictly defining the static functionality of a smart contract on a blockchain, thus automating the acceptance of terms and conditions of a smart contract upon usage of it due to the cryptographic signatures involved. This is especially useful in dynamic situations were multiple parties must affirm the validity of a contract in a limited timeframe, greatly speeding up the process.

To properly gauge the relevance of Ricardian contracts in the LLs of ICONET, the LL operators were reached out to and consulted for validation of use cases. This validation process also included the outputs of version 1 of the deliverable, which was slightly abstractive in the way the solution was formed. The feedback from the ICONET LLs was then combined with existing content to produce a satisfactory workflow the software of version 1 and version 2 of the deliverable.

Once preliminary mapping of the LLs with a Ricardian contract use case was finished, the developmental steps taken to carry out the solution were listed to provide insight to how the technical approach of the software solution was conducted. Numerous researches & legal papers were consumed to estimate to what degree a Ricardian contract can be dynamic and the best approach at implementing this trait. The resulting software was then once again brought forth to the LL operators and discussed to create a LL workflow that will satisfy their needs and produce numerical and non-numerical KPIs to measure the effectiveness of the solution. To conclude, the work ahead that will be included in the upcoming version 3 of the deliverable is elaborated to provide insight to the course of this Task's fulfilment.

Chapter 3 pinpoints the version 1 essence that is pertinent for Ricardian contracts and provides a brief introduction into the ICONET blockchain's purpose. It is important that the reader possesses a clear picture of what ICONET blockchain related facts are taken for granted during the authoring of this paper. This includes any material from the previous deliverable that is to be updated to suit the purposes of Ricardian contracts.

Chapter 4 devotes itself to illuminating the practicality of Ricardian contracts from both a generic and a PI-eccentric point of view. The desirable traits and positive effects of Ricardian contracts are highlighted with the accompanying smart contract functionality being defined and closely associated with them. These effects are then accommodated to ICONET's LL specifications following alignment with the LL operators.

Chapter 5 constitutes of the activities relating to the development of the source code for the software that will support and materialize the expectations of the previous chapters. These activities are taken with legal and academic material into account to ensure that the legal enforceability of ICONET's Ricardian contracts is not diminished due to their dynamic nature.

Finally, Chapter 6 represents the joint effort between the LL operators and the authors of the paper to devise a LL use case scenario that satisfies the needs of the LLs and showcases the usability of the Ricardian contract-based solution. These use cases are accompanied by measurement tools that produce KPI in the form of numerical outputs and can be used along with the included theoretical analysis to estimate the usefulness of the produced software.

The report concludes with eight (8) Annexes that include supportive material for the automatic generation of Ricardian contracts: Annex I: D2.9 Blockchain Specifications, Annex II: Blockchain Source Code, Annex III: RESTful Server Source Code, Annex IV: Full Test Suite, Annex V: Benchmark Suite for LL1, Annex VI: Benchmark Suite for LL2, Annex VII: Benchmark Suite for LL3, Annex IIX: Benchmark Suite for LL4.

Overall the approach taken in this report attempts to be as abstractive as possible, since more efficient blockchain solutions are likely to emerge in the coming years and the PI requirements and use case scenarios are fluid.

3 Precedence

This report follows an earlier document that provided details regarding the ICONET blockchain implementation as well as indicative use case scenarios that were then mapped to ICONET's LLs. The minutiae of the various technical components involved in the ICONET blockchain are crucial to enable the reader to better grasp the scope of this document as well as the terms contained within.

Certain supplementary data that was contained in the previous report has been stripped from the following chapters to maintain clarity and keep the text informative yet summarized.

3.1 Blockchain Solution

As concluded in the preceding report, the blockchain of choice for the ICONET project is a purpose-built blockchain developed under and for ICONET by utilizing Tendermint as the backbone. The blockchain itself supports a very specific instruction set built for the purposes of PI and specifically for establishing and maintaining SLA terms on the blockchain itself. Thus, the system that is composed by these instructions can be appropriately called a "smart contract".

The API that interfaces with the blockchain will also be exploited in and expanded by this report to support the newest set of instructions as developed in the forthcoming chapters. This API conforms to the RESTful API format and has dedicated subscribe endpoints for notifications by the blockchain in case an SLA term is broken. These endpoints were created by deriving methodologies from the SELIS Pub-Sub infrastructure.

3.2 Smart Contract Implementation

At its current state, the blockchain supports the tracking of X and Y coordinates on a Cartesian plane and uses the Haversine formula to calculate the speed of the tracked object. If that speed exceeds a previously imposed limit, the term is labeled as "broken". All these actions are directly processed on the blockchain itself to ensure that the calculations are correctly carried out by all parties.

Within this version of the document, we expand the blockchain implementation to support the tracking of other types of SLA guarantees, such as temperature and humidity guarantees, for a designated tracking. The blockchain's internal structure will also be updated to reflect the association of a specific object tracking & limit with an accompanying Ricardian contract via hashing algorithms.

4 Ricardian Contracts

The Ricardian contract term was conceived in 1998 by Nick Szabo[1] but has since faded into the background as it was a highly innovative design paradigm for legal contracts back then and the market was not yet ready. As blockchain technology came to light, Ricardian contracts were once again studied[2] and brought to the spotlight due to their effectiveness at strictly defining a set of instructions that are conducted by a piece of software and legally represent them under the law. Specifically, blockchain technology companies like EOS have instructed smart contract developers to utilize Ricardian contracts as a robust legal security practice.

Due to their nature as a purely textual document[3], they can come in any readable form and as such enable them to be both human- and machine-readable. This characteristic is important for the purposes of ICONET as it enables us to read from, make changes to and then store Ricardian contracts programmatically. Ricardian contracts have been used and presented in legal courts[4,5] so their enforceability, provided that they are not tampered with by an automatic method, is proven.

4.1 Usability

What matters most for ICONET is whether Ricardian contracts can be properly used with purpose-built blockchain implementations rather than purely smart contract implementations. Preliminary investigations into the usage of Ricardian contracts indicated that Ricardian contracts are not limited by the technology used but rather by the way it functions. The glossary of a Ricardian contract enables it to describe any past, current and future technological functionality as it allows arbitrary usage of the English language.

These results were upheld by further in-depth searches in previous usage cases of Ricardian contracts. These use cases range from financial instruments to auction-style marketplaces, showcasing a wildcard application area. The reliability of a Ricardian contract can be strengthened if it is either reviewed or concocted by a lawyer specializing in its application field. Although this type of strengthening is non-mandatory, it is highly advised for enterprise-grade applications.

As Ricardian contracts are legally binding, they can enable either party of the agreement to raise a court case in case of a dispute of the terms contained therein. The geographical validity of the Ricardian contract is unconstrained provided that the text contained within the Ricardian contract can be digested by the courts of law within the geographical area. However, Ricardian contracts are not restricted to any specific subset of the English language and as such one may adapt a Ricardian contract to suit the geographical context he or she wishes.

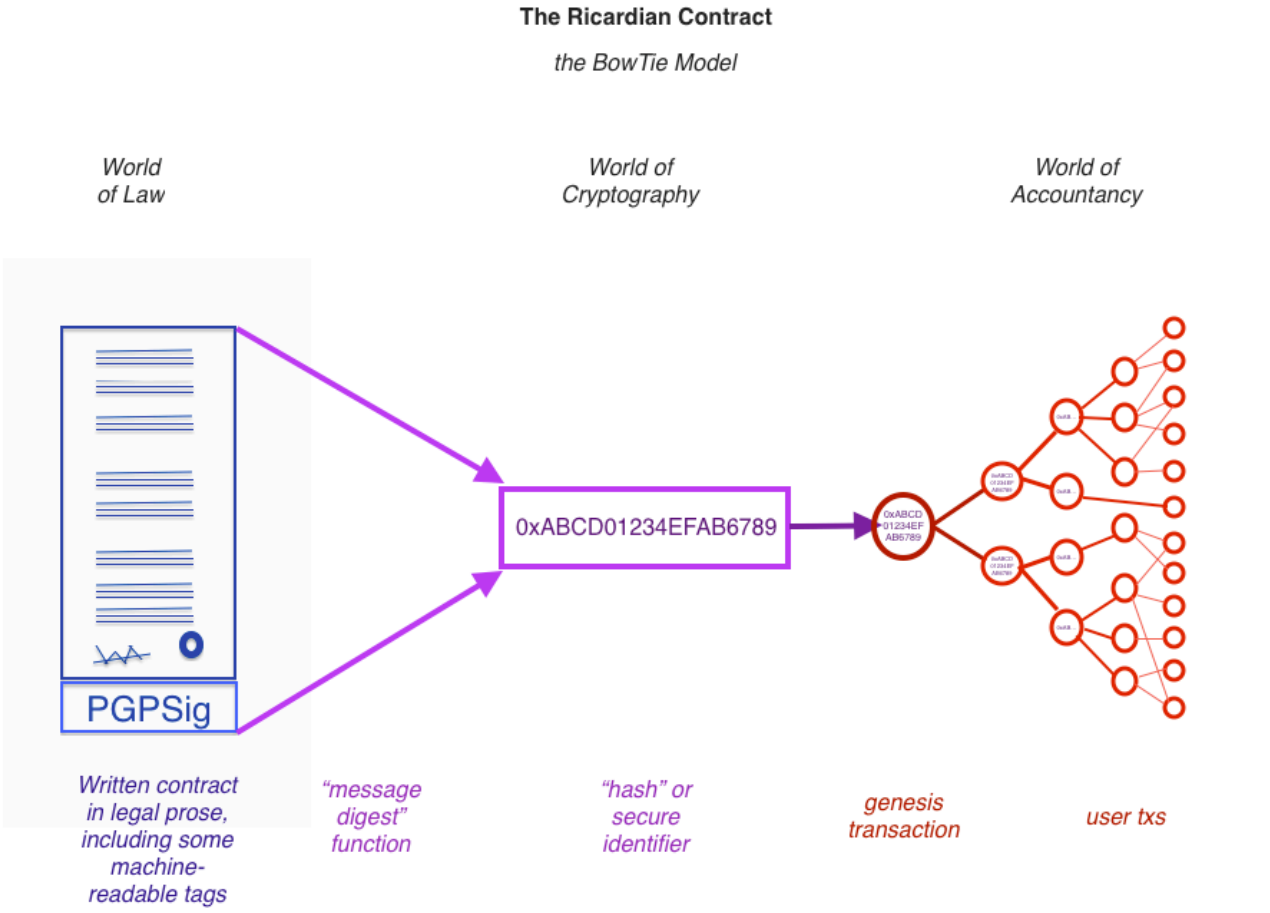

Figure 1 – Bow-Tie Ricardian Model

Additionally, certain Ricardian contract design paradigms can be exploited to further expand the area of effect of such a contract. For instance, the Matryoshka Model, or Russian Doll Model, which is an evolution of the Bow-Tie model allows Ricardian contracts to reference previously valid contracts and mutate them by adding terms, voiding previous ones or editing existing ones. This model has already been applied in a production environment by OpenBazzaar & Open Transactions[7] and has shown to be especially useful in situations when certain types of transactions occur repeatedly.

An immediate deduction of that is that the same model could potentially be useful for ICONET as well, as certain types of transport agreements will see themselves repeated across the industry regardless of the parties involved in the transaction. As an example, a template Ricardian contract regarding the transport of a container from point A to point B under preset conditions will see itself replicated across the PI network, the only difference being the diverse preset conditions.

4.2 Smart Contract Correlation

Ricardian contracts are usually associated in a one-to-one fashion with a smart contract, although it is possible to associate more than one Ricardian contract for the same smart contract as long as different sections of its functionality are described. An area where multiple Ricardian contracts for the same smart contract would be meaningful is the sale of tokens and the tokens themselves, as one could split the legislative description of a tokens sale with the legislative description of a tokens functionality and have users accept the corresponding Ricardian contract when interacting with the smart contract.

To properly ensure the acceptance of a Ricardian contract when interacting with a smart contract, a precondition on the smart contract must exist that prevents the user from transacting with it unless the user has accepted the terms set forth by the relevant Ricardian contract. This is usually achieved by setting up a dedicated function on the smart contract that associates an identifier of a user, usually his public key, with the acceptance of terms by a Ricardian contract based on a cryptographic hash of it that will be signed by the user. The process can be taken care of seamlessly by hiding the complex interactions behind an intuitive interface and display simple “I Agree” or “I Disagree” buttons in the UI of the user.

While not strictly a requirement, it is expected that the hash of a Ricardian contract as visible on the smart contract can be easily associated with its contents from an external user, either via provision of the smart contract’s Ricardian contracts via a web-interface or via direct blockchain linkage with an accessible URI or URL. By empowering the user to verify a contract’s terms on his/her end, we guarantee that the user has knowledgably agreed to a set of terms.

The terms used to circumscribe a smart contract are identical with any other digital process as, contrary to popular belief, smart contracts do not constitute alienated technology. Its functions can be stringently delineated, even more so due to the code’s existence on the blockchain, and the contract’s immutability means that any revisions of the codebase will abrogate the Ricardian contract forthwith. This showcase the optimal interoperability between Ricardian and smart contracts, as the immutability of smart contracts pairs nicely with the austerity of Ricardian contracts.

4.3 LL Mapping

The ICONET project possesses a total of four sui generis living labs that pursue to assess and strain the ICONET system of services and produce measurable and immeasurable indicators of its efficiency and applicability. For us to declare the Ricardian contract aspect of the blockchain complete without consideration of these testbeds is ill-considered at best lest we wish to develop a feeble solution. The following sub-sections enumerate the LLs and exhibit how each is complemented by the ICONET Ricardian contract solution.

4.3.1 LL1: Hub Network

Under the premise of the first LL, the port of Antwerp will be studied, and the pros and cons of a PI hub network established within it will be gauged. The high inflow of shipments renders it prime ground for the appraisal of the ICONET Ricardian contract solution, as the legal contracts that need to be established for each shipment among shippers, deep sea carriers, freight forwarders, railway undertakers and terminal services providers amount to a considerable operational overhead both in fiscal and timely terms. The elimination of the human factor from the contract’s composition process lessens room for error and depresses this overhead.

The Matryoshka Ricardian contract model can also be applied on this LL efficiently as it allows the abstraction of a typical transport contract and its swift repeated usage. Linkage of a smart contract representing a specific SLA or more with a Ricardian contract will occur at the contract level whereas parameterization of a Ricardian contract will allow automatic deployment and correlation.

4.3.2 LL2: Routing Optimization

The narrative of the second LL pertains to the optimal route calculation for PI packets circulating in the PI network and its accompanying PI corridors by utilizing Value Stream Mapping (VSM) in conjunction with the route optimization algorithms of ICONET. Key performance indicators of this LL Is the reduction of its environmental impact, the increase in SLA conformity and the general improvement of the overall system in comparison to non-intermodal systems.

The LL is a natural fit for the Ricardian contract-based SLAs, as they can be autonomously enforced whilst the packet is in transit rather than at checkpoints throughout the transport route. Furthermore, the speed between route calculation and transport initiation will be significantly reduced as the surplus overhead of writing and manually signing legal contracts will be replaced by the digital contracts of the ICONET blockchain. The environmental impact of the LL as a whole will also be contained, as the physical paperwork surrounding the agreement of a certain PI route will be eliminated. This metric can also be measured in grams of paper and printer ink avoided by measuring how much paper and ink each blockchain contract would consume, thus providing tangible output as to how efficient the system is.

This LL also highlights the ability of the blockchain to consume IoT measurements to be fed to the Ricardian contract-based SLAs, as these events will pass through the Web Logistics service which will interact with the blockchain and store the data feeds where applicable for actuation when necessary.

4.3.3 LL3: Dynamic Situational Routing

The third LL possesses the same narrative as the second one, as it concerns the dynamic routing of smaller PI-containers within a PI network comprised of an extensive complex network PI hubs (SONAE central warehouses, dark-stores and convenience stores), albeit it takes into account the situational context as well, such as whether certain roads are or have been blocked off during transit. Once again, Ricardian contracts can act as a tool that streamlines the process of legally establishing the new dynamically calculated route on both the transporter's and the receiver's end. These contracts could be formed between the Shipper and either the various carriers involved in the transportation of the goods or the PI-hub owners, ensuring the network's capability of dynamic expansion on demand, as these processes would be automated and facilitated by the blockchain.

As this LL also concerns dynamic routing in-transit, the ICONET blockchain and Matroyshka model can highlight the ability of amending a contract's terms and adjusting the comparison parameters for the SLAs, such as the GPS coordinate range that is acceptable in case of a new path. This is especially useful in the context of dynamic PI as current pen-and-paper SLAs can't be adapted and updated quickly according to situational changes.

4.3.4 LL4: Warehouse Capacity Allocation

The final LL attempts to digitize the physical logging of warehouse storage availability to breed a variety of new and useful services to life assimilating this newfound digital trove of data. These services include the dynamic orchestration of where a PI packet should be stored during its travel path depending on which PI warehouses it will pass by and their associated current and projected availability.

Here, the ICONET blockchain can play a pivotal role in enhancing the digitization process of the warehouses by introducing a means of automating the contractual lease of warehouse space to an entity and the severance of such a lease. This can open up new, fully autonomous ways of arranging packets in the PI without requiring human interaction. In the case of Stock booking, it would empower the entity to form new agreements with warehouses outside their network in a straightforward way with minimal effort from either side.

In the future, the blockchain solution could also encompass any types of events useful if associated with a particular warehouse, such as the storage of goods under specific environmental conditions e.g. temperature / humidity. This would enable an auditable trace of the inner state of a warehouse without necessarily revealing the state publicly to other actors in the network and instead being revealed on a need-to-know basis.

5 Programmatic Generation of Ricardian Contracts

To accord with the needs of the progressive setting of PI, the Ricardian contracts that will be defined and referenced to within the ICONET blockchain need to support some form of dynamicity, otherwise they would be unsuitable for the purposes of ICONET. This proves to be a complex predicament, as the legal enforceability of Ricardian contracts after they have been altered via programmatic methods may be rendered void.

Within this Chapter we investigate the possibility of automatically generating modular Ricardian contracts and / or using Ricardian contract templates, with the possibility of defining a Ricardian contract explicitly in the worst-case scenario. Preliminary research indicated that it is indeed possible for Ricardian contracts to be automated, or rather "automatable"[2]. We further develop the initial research conducted during the project's conception to solidify our findings and transition them into actual code that will conform to these findings.

5.1 Legal Enforce-ability Investigation

The idea of smart contracts, in their definition as legal contracts in machine-readable digital form, is a relatively new concept that received renewed momentum with the advent of blockchain technology due to the immutability and auditability of blockchain smart contracts pairing nicely with legal smart contracts. Although the most renowned type of legal smart contract is the Ricardian contract, other initiatives that attempt to simplify the structure and meaning of legal prose in machine terms have been formed such as Legalese[7,8] and CommonAccord[9].

The former wishes to create legal smart contracts by defining a DSL, L4[10,11,12], for legal agreements and regulations that is multilingual and can be parsed by machines to create meaningful text and data for end-users. This initiative tackles modularity via its DSL, which basically allows any combination of words and therefore legal prose as long as it is compliant with the syntax and lexicon of the L4 DSL. It is currently offered as a paid SaaS that handles the full lifecycle of a contract such as the signature process, revocation procedure, amendment workflow etcetera.

The latter focuses more on a standardized approach that allows contracts to co-exist in their own language with a common lingua franca for global legal transacting. It ventures into the area of smart contract templates by expanding upon priori research[3] to conclude that smart contract templates can exist and combine both automation and conventional legal text to ensure their compliancy with law via codified prose objects.

The findings of the aforementioned papers align with the assumptions made during the conception of the project and as such allow us to proceed with the enactment of legal contracts in a digital format. In our case, we assimilate the findings of these initiatives and enhance them when necessary to properly fit the narrative of the ICONET project. In detail, we adopt the methodology of the CommonAccord, which is associating an actual legal document with a digital Ricardian alternative and segregate the logical blocks that make up the contract to allow one to define a contract using pseudo-commands akin to how a DSL operates, as is the case with Legalese.

Figure 2 - CommonAccord Ricardian Contract Illustration

By combining the best of both methodologies, we are able to devise an easy-to-use system designed specifically for PI and have it been fully compliant with the specifications of Ricardian contracts as well as most judiciary systems, as long as the underlying legal contract that supports the Ricardian one fulfills the necessary requirements of the judicial area it wishes to be valid in.

5.2 Identification of Candidate Legal Contract

In order to deduce and devise the candidate legal contract to be transformed to a Ricardian contract for the purposes of ICONET, we needed to gather intel as well as specimens from the various LLs. We reached out to each LL's host and accumulated the documents that were at their disposal to provision how we envisage a singleton transport agreement document to satisfy the criteria and terms laid forth by the acquired documents.

From hereafter, any mention of a "user" relates to a user of the API endpoints offered by the service and not the actual PI users themselves. Additionally, to avoid repetition, it should be noted that apart from the "manual" definition of parameters in the API endpoints from the "user", the blockchain service is capable of parsing a sample regular contract, extracting the info it is interested in and converting it to a Ricardian contract. This conversion was included in the prototype version of the blockchain service created for ICONET, however it should be noted that the service itself acts as a guideline to how one should create a consumption endpoint and create the Ricardian contracts.

The system itself is easily extensible and allows "users" to upload their own Ricardian contract templates to the platform, thus ensuring that the ICONET blockchain solution will be prevalent in the future. The conversion methodology while provided in the ICONET project is not the essence of the Task's research output as it is a broad subject that is practically impossible to accommodate to in full. The goal of the Task's research output is to define a methodology via which one is capable of forming and utilizing Ricardian contracts in the PI context to replace traditional contracts.

Scope of Services It is important to have a clause describing each service that you will be offering to your client, including those that come with additional charges. A detailed list of services can help prevent possible controversies in the future. Standard of Care The standard of care that you will be providing for your client should be affirmatively defined. While you should not describe your standard of care as "perfect," it should be high enough to meet your client's expectations. Termination There should also be a termination clause in the contract that describes the circumstances under which the legal relationship can be terminated. Contracts that never expire can be problematic after some time. The best thing to do is to allow renewals at certain intervals. Third-Party Beneficiaries The transport contract must also address the possibility of third-party claims, which are claims of individuals and entities outside the contractual relationship. You can form ground rules for court mediators to use for guiding interpretation. Rubber Stamps You should not just ask an attorney to go over the transport contract and approve it just because you have been working on it for a month or so. Having a lawyer involved in the early stages of the drafting process can enhance your leverage and well-being.

5.2.1 Billing and Payment

The first section that was shared between the transport contracts were billing and payment clauses. Evidently, the contracts limned the methodology via which the purveyor of the transport service would reimburse the provider of it. Within it, explicitly expressed sums were detailed in unambiguous legal annotation. The payment method was also specified along with clearly written internationalized date deadlines in accordance with the fulfilment of the services, e.g. at successful delivery.

Supplementary information that was also defined in this section were any form of late payment penalties for the purveyor of the service, such as accumulatory interest, collection costs and any legal fee reimbursement in the event of a dispute.

Digitizing this section of the document requires intricate care on how the various dynamic variables of it are properly parsed and transformed to the internationalized and unambiguous formats demanded by legal prose. While the fiscal values and method of payment are easy to replace dynamically in a document, the date of payment required slight adaptation.

We identified a common pattern between the contracts analysed, and that was that the date of payment was always specified relative to the completion of the provision of those services. As such, we created a legged format for specifying the time of payment relative to delivery supporting the following time units: hour(s), day(s), week(s), month(s).

5.2.2 Certifications and Warranties

Here, any special warranties that need to be made for the transport of an item are laid out along with any certifications that need to be adhered to. For instance, if a temperature-sensitive item is transported the temperature ranges it will be kept in will be detailed here. The documents we analysed showcased that almost all transport services operate on a best-effort basis, avoiding the mention of any explicit pickup or delivery time warranties or percentage of customer satisfaction. This is a commonly accepted fact in the industry, as transport services are heavily affected by numerous entropic external factors such as environmental conditions.

The types of warranties regarding the transport conditions of an item are very vast and redundant for the scope of the ICONET project. Instead, we decided to focus on supporting a few of these types of warranties to encompass the needs of the LLs. Those terms were temperature ranges and delivery times that are shorter than a product's lifetime. These two types of warranties were again split into logical blocks that can be included at will and their parameters can change e.g. the upper and lower temperatures an item can withstand.

5.2.3 Consequential Damages

This section shared common segments between all contracts analysed and essentially describes the waiver of any consequential damages that may be incurred from loss of profits or similar indirect costs that may result during the transport process. The waiver's existence is paramount for the legal protection of transport service providers against baseless damage claims.

As the consequential damages clause mostly included textual content that cannot be reasonably parameterized and converted to a sentence, we have implemented a default representation of the consequential damages clause and allowed the user to potentially include their own interpretation of this clause within the resulting contract. Delicate care should be taken when conscribing this section of the document, as ICONET does not parse and analyse the actual legality of the content provided in this section.

5.2.4 Safety

Once again, this section of the document limits the liability of an employer and purveyor of services in case of a work-related accident and ensures that the appropriate work-safety measures will be taken when carrying out the transport of an item. This clause clearly defines that the responsible party during transport is the individual or superseding entity that fulfils the task and prevents potential litigations and lawsuits in the unlikely and unfortunate event of a work accident during transit.

This section appeared relatively standardized and as such, we simply integrated it into the contract and provided the user with an opt-in feature with regards to its inclusion in the final contract output.

5.2.5 Liability Limitation

The purpose of this section is to limit the weight of liability a transport operator possesses in cases and claims arising regarding the service, such as the theft of an item or the item's destruction during shipment. We perceived that this section usually defined the liability in percentage of the total fiscal impact of such a claim.

To illustrate this section properly within our generated document, we once again allowed the users to provide valid numerical values with up to two-digit decimal precision to accommodate for any peculiar cases of liability limitations.

5.2.6 Alternative Dispute Resolution

This section was not widely prevalent within the documents analysed but should be mentioned as it ties in nicely with the blockchain-based solution we provide. It essentially designated an alternative method of resolving disputes via the usage of a mediator that will investigate the dispute and provide an affirmative or negative response with regards to the validity of the dispute.

In the case of ICONET, we can include this section as opt-in to enable future support of the ICONET's built-in blockchain-based dispute resolution mechanism. This resolution mechanism will be explored and prototyped in version 3 of this deliverable, D2.11.

5.2.7 Scope of Services

A detailed scope of services was provided in each of the documents we consumed and offered meticulous descriptions of the said scope as well as any surcharges that accompany their provision. This section is included in the contracts to avoid any discrepancies or controversies whilst the service is still being carried out.

Tantamount to how we approached the consequential damages clause, the ability of inserting their own interpretation of the scopes of services section is offered to the user.

5.3 Smart Contract Implementation Details

The purpose of the ICONET blockchain service is to create a smart contract counterpart of any legal contracts that enter its network. Our initial approach dictated that the users of the service possess the contracts they wish to convert to the Ricardian contract format we broke down above. This is a naïve assumption that will not be met if we wish to apply the ICONET methodology in a real-life scenario, as all the parties of the PI network would not allocate the appropriate amount of resources to construe these Ricardian contracts and convert their traditional ones to Ricardian ones.

To solve this, we have devised an API endpoint that allows the generation of Ricardian contracts via machine-readable document input. The schema of the machine-readable document input the endpoint expects conforms to the specifications of the ICONET Architecture (D2.1) and Protocol Stack (D1.10), where handling instructions are mentioned within the requiredHumidityRange property of the product object as an example. The property example was taken from the upcoming version of the Architecture deliverable. In the end, the generated Ricardian contract simply needs to be associated with a smart contract for it to be enacted in full.

The association step needs to occur on the blockchain itself rather than an external function or service as there needs to be tangible and conclusive evidence that a smart contract represents a particular Ricardian contract in a 1-to-1 fashion. Storing arbitrarily long Ricardian contracts on the blockchain is an inefficient task and one that waste resources as there are much more efficient ways of linking a large dataset with a small identifier. For this purpose, we utilize a hashing algorithm to generate a unique alphanumeric sequence that will represent a Ricardian contract.

We utilize the SHA-2 256-bit algorithm for generating the hashes and we expose a one-time callable function on each smart contract deployed for a Ricardian contract that allows the storage of the hash to the smart contract's permanent memory. This enables anyone to verify a smart contract represents a specific Ricardian contract provided he / she has access to the Ricardian contract's original text to produce the corresponding hash.

As a next step, we will adapt the smart contract functionality created in version 1 of this deliverable to better reflect the needs of the LLs based on the provided contractual documents. Specifically, we identified that turbulence sensitivity, temperature fluctuations and sunlight exposure are three major factors that most carriers are concerned with during the shipment of a package rather than the speed limit that was imposed in version 1 of the deliverable. As such, we proceeded to alter the codebase of the smart contract and accompanying API to support these types of metrics via ranges and simple conditionals.

These conditionals include the following conventional comparison abbreviations: gte (Greater than or equal to), lte (Less than or equal to), gt (Greater than), lt (Less than), eq (Equal to). We believe the ranges that can be defined by combining the aforementioned abbreviations fulfil the full range of possible permitted values and ranges for a particular metric, e.g. temperature.

The concept of PI is relatively immature and with the rapid advancement of IoT technology we anticipate more types of metrics to be introduced in the near future and carriers to potentially adapt their contracts to reflect SLAs for these new metrics. To accommodate this potential need, one may categorize the types of metrics whose SLA can be described by a range as f.e. range metrics. This enables one to create identifiers for these metrics dynamically and listen to them via a dedicated consumer endpoint on the ICONET blockchain interfacing API.

By specifying unique identifiers in the form of a pseudo-language, it would enable parsing of an arbitrary-content contract as long as the correct variables are marked using the corresponding identifiers. This solution would allow one to basically parse a wide variety of contracts with minimal work from the contract owners.

As an example of a templating pseudo-language, let us say we have a new contract whose Ricardian contract we wish to generate. We define an identifier in a similar fashion to how template literals are used in the JavaScript ES6 programming language. Specifically, we expect to find the metric's name enclosed in brackets ({}) and preceded by the dollar sign ($). Take the below sentence as an example:

The PI packet shall not surpass the ${temperature} ranges set forth by this agreement.

This allows us to extract the "temperature" as a metric identifier and listen to events relating to the "temperature" of a packet. The range of the permitted temperature values can be identified in many ways. One would be to define the range within the template literal, another would be to apply neural language processing techniques and skip the templating pseudo-language step entirely by processing the actual contracts and deriving the corresponding metrics and limits from it in a programmatic way via contextual analysis.

As proven by the above two methods, it is possible to generate Ricardian contracts and extract the corresponding metric SLAs in a programmatic way thus rendering the ICONET blockchain solution future-proof. However, the NLP and general templating aspect described above is out-of-scope of the ICONET blockchain task whose main purpose is to define a methodology to create Ricardian contracts and associate them with smart contracts. We outlined the methods above as potential enhancements for the project post-completion.

As mentioned earlier, the materialized API of this deliverable expects machine-readable input as specified by accompanying deliverables of the project's architecture. We can still support the category of range metrics dynamically within our Ricardian contract, provided that the associated legal document block for that particular metric has been created. For ICONET, we have created the legal document blocks for temperature, turbulence and sunlight exposure metrics allowing us to cover the full breadth of expectations set forth by the LLs.

We assume that the machine-readable input will also include a unique identifier for the PI packet we wish to track the SLAs for, to properly filter the events we receive on our event endpoint. The smart contract, once linked with a Ricardian contract, will enable its full API to any external party that wishes to interact with it. As blockchain technology is intricate and not commonly understood by all, we expand the blockchain's API to support the API of the underlying smart contract in a 1-to-1 fashion to allow users to interact with it via a conventional RESTful API.

Underneath, the smart contract exposes a function that will allow the assignment of an event's values to the history of a PI packet and verify that the metric falls within the permitted value range of the Ricardian contract. If the values reside outside the permitted values of the Ricardian contract, an alert event will be raised and parties who have subscribed to SLA severances in the SELIS inspired API developed in D2.9 will be notified with a timestamp of the severance moment. This will allow listeners to propagate actions where necessary to either halt the transport of a PI packet, return it to the sender and / or initiate renegotiation procedures for adjuster reimbursement rates.

As the full history of recorded metrics will be processed by the blockchain, it is necessary to prune it once a transport agreement has been completed to avoid burdening the blockchain with unnecessary data and thus rendering it incapable of performing under heavy load. This form of selective pruning will ensure the blockchain's favourable performance when deployed in a real use-case scenario.

What we have described thus far are the implementation details for a smart contract relating to a single packet. To avoid code repetition on the blockchain, we utilize a singleton contract instance that will be able to support multiple packets and multiple contracts. To do this, during initialization the contract needs to initialize the arrays that will hold the contracts, corresponding packet ids and limits and subsequently expose a function that will allow the inclusion of a new contract and packet to the blockchain.

To maximize the efficiency of the system, we offload whatever functionality we securely can to the overlay API to ensure that the blockchain acts as a catalyst only for the subset of instructions that must be executed in a securely and auditable manner. For instance, the transformation of a JSON event to a blockchain-consumable event does not necessarily need to be done on the blockchain as it would waste system resources otherwise useful for more important instructions such as the auditable storage of an event's values.

The complete code of the smart contract and accompanying RESTful API can be found in Annexes 2 and 3 respectively. These annexes also include links to the repositories that contain their code. To enable ease-of-deployment of the blockchain in the LL scenarios, we have also coded a Docker container via a Dockerfile that spins up an instance of the blockchain and attaches the RESTful API to it, allowing any partner to easily deploy and interact with the ICONET blockchain.

5.4 Correlation to the OLI Layers

The main goal of ICONET is to develop a technological solution that will apply to the PI domain and sit within the OLI model. Over the course of the 2.1 deliverable, a workflow was defined that indicates where each service of the ICONET project sits in the OLI layers. This workflow, which is depicted below, includes the blockchain component of ICONET and sets it in a pivotal role, acting as the instigator of the whole process after the first contact with the freight forwarder:

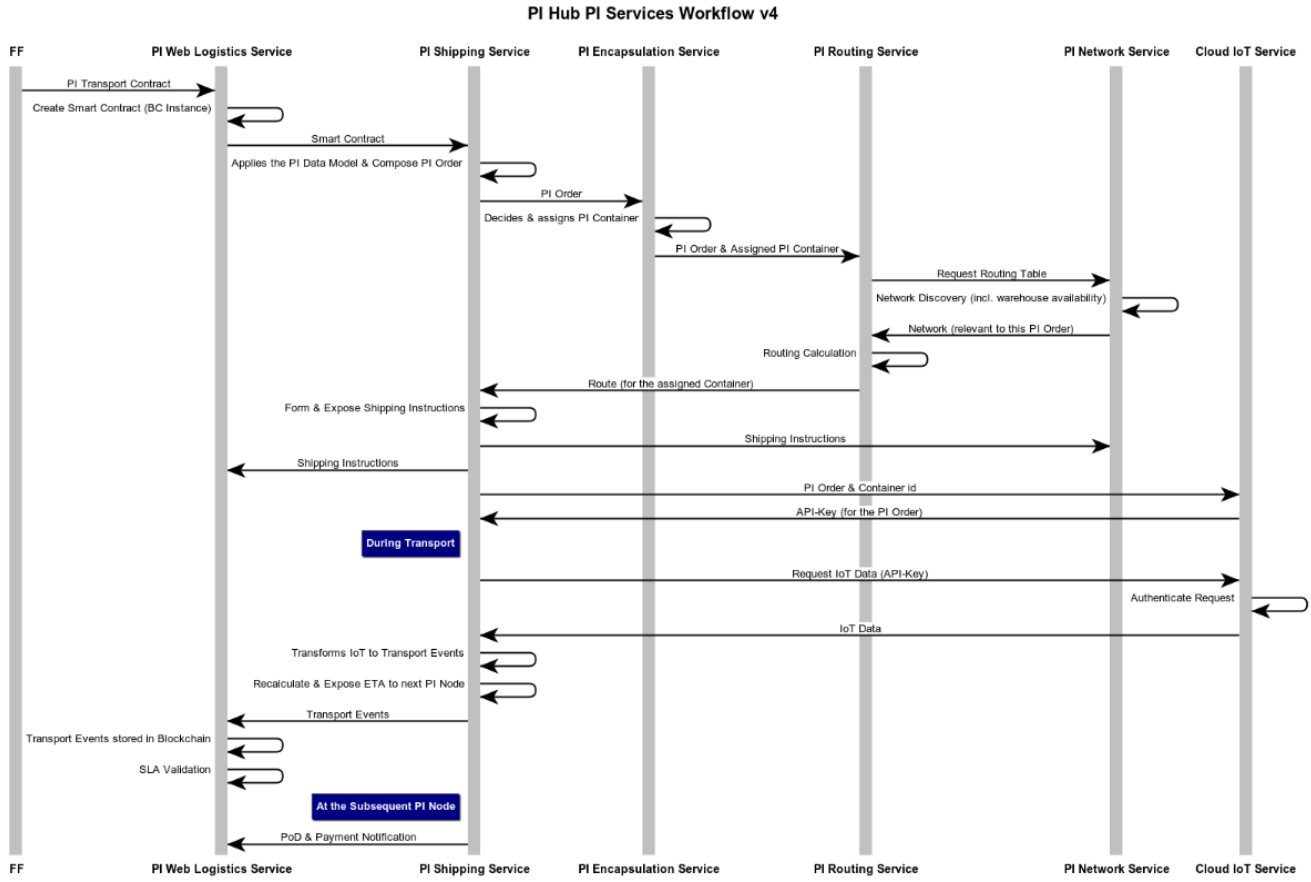

Figure 3 - The ICONET Services within OLI

The RESTful API developed in this deliverable acts as the PI Web Logistics Service for the OLI layers with the aid of the blockchain instance spawned within it. As indicated here, numerous interactions occur between the PI Web Logistics Service and other actors in ICONET. Breaking down the diagram in detail, we initially expect to receive a PI Transport Contract in our service which will subsequently form a blockchain smart contract within the blockchain that will represent it. We expect the PI Transport Contract, according to D2.1, to take the form of an Order in the ICONET Domain Model and to boast the following format:

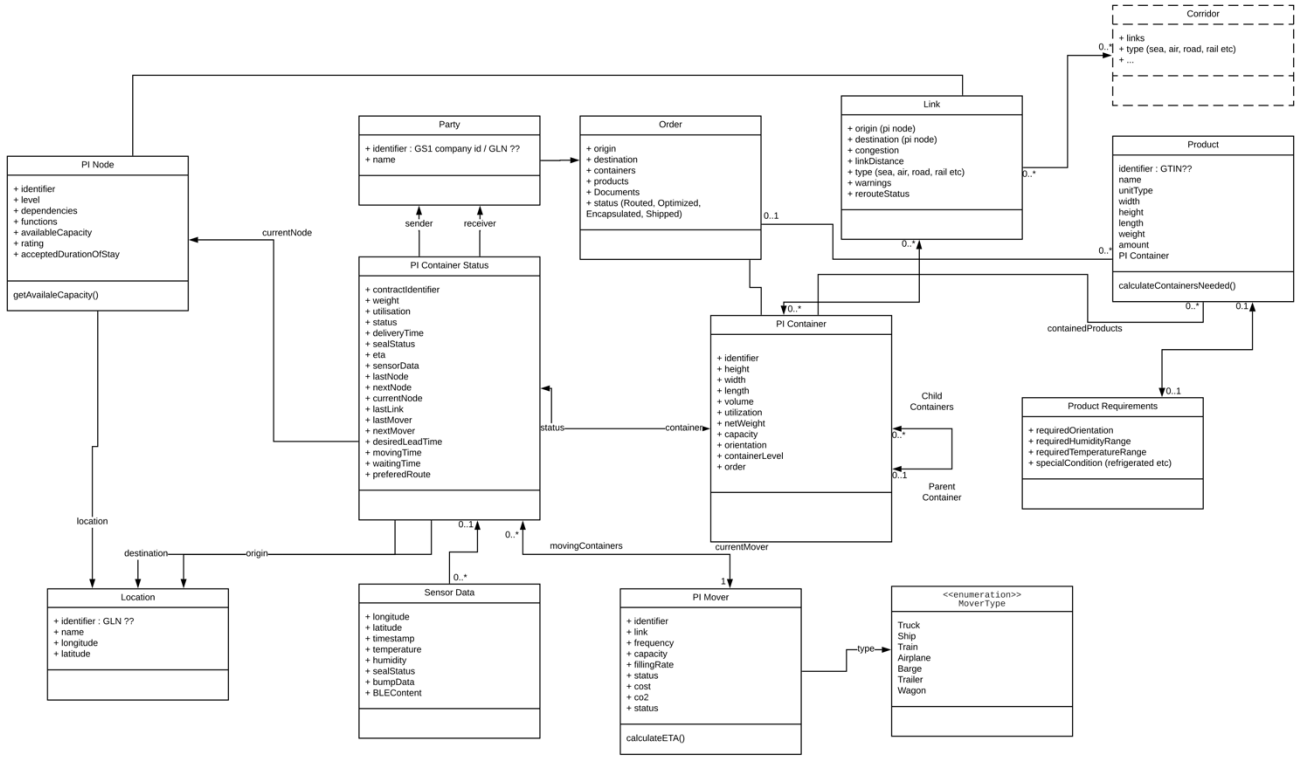

Figure 4 - ICONET Domain Model

In technical terms, we expect to receive a JSON payload with the attributes defined in the Order entity above where we will iterate through the one-to-many relation between the Order and PI Containers and create individual smart contracts for each container. With regards to the SLAs that need to be kept per container, we also iterate through the products attribute of the Order entity and attempt to find a Product entity that refers to our PI Container and derive the limits from it. This payload is meant to be delivered as a POST HTTP request to our /contract endpoint, as defined in Annex III, the API source code.

For subsequent interactions with the blockchain contract, we have devised the corresponding RESTful endpoints all interfaced to via PATCH HTTP requests. As an example, recording IoT device measurements to the blockchain and verifying them with the corresponding SLAs occurs via the /contract/:id/metric endpoint which expects a JSON payload in the form of an attribute named metric and a numerical value. The :id in the above URL path is meant to be swapped with the identifier of the container the measurement is directed to.

For fulfilling the functionality of the other arrows in Figure 3, we expect a request to the /contract/:id/event endpoint which consumes a JSON payload in the form of an attribute named event and an arbitrary structure object. This arbitrary structure object possesses a single reserved keyword, an attribute named final, which is represented by a Boolean value and, should it be true, marks the event as the final one in its transport path and is meant to be used along with an event signalling the Proof of Delivery and Payment notification in the Diagram of Figure 3.

6 Full Workflow Showcase

To measure the impact of the ICONET Ricardian contract blockchain solution, we measure the performance of the blockchain under a set of edge-case conditions to validate that it operates correctly as well as generate metrics that can be used to calculate the theoretical savings a PI party would procure if they integrate the ICONET blockchain solution. For this particular purpose, we measure the page length of the generated Ricardian contracts as well as the median paper and ink cost for the print of a single page to calculate indicative total savings in monetary value of the ICONET Ricardian contract blockchain service.

The dedicated per-living lab tests will utilize the same TDD framework that was utilized in D2.9 whereas the benchmarking tests will utilize autocannon, a library that aids in stress-testing API endpoints. The results of these tests will be analysed to produce measurable outputs that will aid us in gauging the impact of the ICONET blockchain solution. Additionally, certain logical indicators will also be explained that cannot be measured, such as the overall security of the system.

6.1 LL1: Multiple Temperature Sensitive PI Packets

In this scenario, we envision that the PI Hub of the port of Antwerp expects a high influx of temperature-sensitive containers which it will need to create contracts for. We have analysed the 2019 maritime cargo turnover of Q1 through to Q3[13] that was provided to us and extracted the relevant metrics, specifically the number of containers that were processed by the port. These numbers amount to 104.072.099 total containers, with 46.091.627 being incoming and 57.980.473 being outgoing. If we assume each container is tied to a contract in a 1-on-1 fashion, 104.072.099 contracts would need to be generated. For a typical Ricardian contract, a minimum of 4 pages of A4 paper are needed and a typical amount of 0.8ml of black ink is used.

Using basic multiplication, we can calculate that the total cost of producing this number of contracts would be 15.610.814,85€ for paper (Cost per 3 A4 sheets: ~0,15€) and 41.628.839,60€ for ink (Cost per 0.8ml: ~0,40€), totalling at 57.239.654,50€. This might merely be an estimation, but it can provide us with insight as to where the actual value of savings will be within a margin of error. Additionally, albeit it is assuming that all contracts would be replaced with digital ones which is an optimistic disposition, this metric also solely includes the material costs, it does not involve the time and manpower cost that would be incurred if the contracts were manually created.

For the benchmarking suite included in Annex V, we created code that carries out the creation of a temperature sensitive PI packet and produces randomly generated temperature metrics to them to measure how the system performs under a life-like scenario. The tables below summarize the test results:

Table 1 - LL1 Latency Percentiles

Stat	2.5%	50%	97.5%	99%	Average	Standard Deviation	Max
Latency	0 ms	0 ms	0 ms	0 ms	0.01 ms	0.08 ms	9.46 ms

Table 2 - LL1 Requests & Bytes per Second Percentiles

Stat	1%	2.5%	50%	97.5%	Average	Standard Deviation	Min
Requests per Second	23215	23215	30927	31167	30262.55	2235.06	23200
Bytes per Second	5.71 MB	5.71 MB	7.61 MB	7.67 MB	7.44 MB	550 kB	5.71 MB

6.2 LL2: PI Packet Turbulence Measurements

For this scenario, the rapid re-calculation of routes and subsequent newly generated contracts will dictate the turbulence margins a packet may be imposed under regardless of the route chosen. This will allow us to test the amendment of the Ricardian contract with an updated one, as well as the retention of Ricardian contract histories and accompanying measurements occurred during the lifetime of these Ricardian contracts.

For the benchmarking suite included in Annex VI we created code that carries out the creation of multiple humidity sensitive PI packets with different metadata each type and subsequently different contract hashes. The tables below summarize the test results:

Table 3 - LL2 Latency Percentiles

Stat	2.5%	50%	97.5%	99%	Average	Standard Deviation	Max
Latency	0 ms	0 ms	1 ms	1 ms	0.04 ms	0.23 ms	15.05 ms

Table 4 - LL2 Requests & Bytes per Second Percentiles

Stat	1%	2.5%	50%	97.5%	Average	Standard Deviation	Min
Requests per Second	11703	11703	15479	15895	15182.91	1121.49	11699
Bytes per Second	3.86 MB	3.86 MB	5.11 MB	5.25 MB	5.01 MB	370 kB	3.86 MB

6.3 LL3: PI Orientation Discrepancy SLA Severance

In this use-case, we will measure the performance of the ICONET blockchain system when a random number of contracts need to be severed and attached listening parties need to be informed. The dynamic situational routing of the PI packets of the LL depends on environmental conditions which may rapidly change and as such we believe it is the ideal LL to test the capability of the system's SLA severance mechanisms.

For the test suite included in Annex VII, we created code that carries out the creation of multiple orientation sensitive PI packets and produces metrics that exceed the permitted range of values set in the contracts. The tables below summarize the test results:

Table 5 – LL3 Latency Percentiles

Stat	2.5%	50%	97.5%	99%	Average	Standard Deviation	Max
Latency	10 ms	12 ms	19 ms	23 ms	12.21 ms	5.4 ms	166.17 ms

Table 6 - LL3 Requests & Bytes per Second Percentiles

Stat	1%	2.5%	50%	97.5%	Average	Standard Deviation	Min
Requests per Second	553	523	818	848	786.37	89.12	523
Bytes per Second	150 kB	150 kB	234 kB	243 kB	255 kB	25.5 kB	150 kB

6.4 LL4: Combination of SLA Severances, Ricardian Contract Alterations & Normal Conditions

The last use-case will maximize the strain on the ICONET blockchain system by combining the test suites above to measure the performance of the system if imposed under heavy load on all fronts. This will include the conditions of the PI packets within the warehouses themselves and ensure all warehouses involved in the storage of a PI packet will conform to the specifications laid out in the SLAs.

For the test suite included in Annex IIX, we created code that carries out the creation of multiple temperature, turbulence and sunlight sensitive PI packets and produces random metrics that can either fall out of or in the ranges defined in the SLAs, with random selective amendments and revisions to the Ricardian contracts of each PI packet. The tables below summarize the test result:

Table 7 - LL4 Latency Percentiles

Stat	2.5%	50%	97.5%	99%	Average	Standard Deviation	Max
Latency	13 ms	21 ms	331 ms	334 ms	41.2 ms	75.31 ms	349.64 ms

Table 8 - LL4 Requests & Bytes per Second Percentiles

Stat	1%	2.5%	50%	97.5%	Average	Standard Deviation	Min
Requests per Second	108	108	261	356	239.4	73.92	108
Bytes per Second	31.5 kB	31.5 kB	76.1 kB	104 kB	69.8 kB	21.5 kB	31.5 kB

7 Conclusions & Work Ahead

The concept of automated legal contracts, while relatively mature, has not yet been widely applied and is considered an experimental technology that is expected to bloom as blockchain technology becomes more widespread. Taking this into account, this report has attempted to delineate a methodology via which rudimentary smart legal contracts, usually dubbed Ricardian contracts, are ex-machina generated and comply with the Ricardian specification. The functionality through which a smart legal contract is generated is relatively straightforward to extend, opening up possibilities for future enhancements such as a wider application area or a narrowed down one. The accompanying software of this deliverable is meant to merely serve as a PoC rather than a full-fledged solution as the currently available tools prohibit the creation of a production-grade solution that attains to the needs of arbitrary contracts. We anticipate that this limitation will be lifted in the future and as such, the ICONET blockchain-based Ricardian contract generation system will be extended to utilize the new tools that lift these limitations.

The functionalities we have pinpointed the blockchain to serve will remain the same regardless of how the technology space proceeds as the 1-to-1 association of a 32-byte string of characters with a particular resource will be an operational prerequisite without any dependency on the method via which the string is generated. Adaptations may need to occur should a new prominent and novel data storage format emerges, such as trytes instead of bytes, however the adaptation will be minor, and the underlying theory will remain the same.

After setting the precedence and outlining the material that was assimilated from the first version of this report, we proceeded to identify eminent use cases of Ricardian contracts by major players in the field of digital law and singled out certain studies that correlated specific digital contract formats with legal contracts and described their findings. We evaluated these studies and sources and deduced that automatically generated legal contracts are indeed achievable and relevant for the case of ICONET.

We then began to define a candidate Ricardian contract that is heavily modular and relies on input by the parties involved in the agreement to be generated. This information was gathered from the D2.1 and D1.10 architectural and protocol stack deliverables respectively to ensure the blockchain solution falls in line with the rest of the services in ICONET. We requested and analyzed exemplary legal contracts from the LLs to target a subset of the terms they contain in common and devise certain LL scenarios that attend to the needs of the LLs and are realistically demonstrable. This subset of terms was expanded in detail to rationalize the choice of the termA distinct LL scenario was devised per LL and is meant to showcase a certain operational aspect of the system as well as strain test the system under close-to-reality situations.

In continuation to the aforementioned definitions, the developmental steps were broken down and detailed to provide an idea of how each function of the contract was coded. The total programmatic output of the deliverable is an extension to the pre-existent RESTful blockchain-interfacing API as well as an extension to the underlying blockchain implementation and subsequently the functions that are exposed therein. An additional, non-blockchain endpoint was also coded to provide the automatic extraction of the necessary variables from the data structure detailed in the upcoming deliverable D2.1 and the programmatic generation of the Ricardian contract defined in this report.

Finally, the outputs of these chapters were conglomerated to ideate full workflows for each of the LL and divulge how the systems are expected to behave under these circumstances. Test suites were provided in code that carry out those workflows and highlight the ingenuity and capabilities of the system were applicable. All the program-related functions that have been mentioned until now are included in the Annexes (V, VI, VII & IIX) of this report and directly linked from their respective paragraphs.

The product of this report is to be consumed by the ensuing version of this report, version 3, to develop a first iteration, if possible, of the automatic arbitration system whereas the tangible services produced by the report will be deployed in the LL simulation scenarios to demonstrate the efficacy of the system.

To summarize the activities of this report, a beta version of the blockchain as an extension to its alpha version was developed that satisfies the functionalities desired by the LLs regarding the automatic provision of legal contracts in a blockchain-enabled fashion. Workflows were specified that map to each LL and achieve the requirements that one would expect from such a solution whilst the KPIs that accompany the test suites of these LL workflows were depicted to enable readers to better grasp the impact of the blockchain solution. These workflows will be integrated with the overall workflow of the ICONET project to devise a single demonstrable workflow that showcases all services of ICONET operating in tandem.

Amendments to the blockchain implementation attached to this report may occur as the project progresses to accommodate for any prospective requirements that may arise in the future from the forthcoming version 3 of this deliverable.

8 References

[1] Grigg, Ian. "The ricardian contract." Proceedings. First IEEE International Workshop on Electronic Contracting, 2004. IEEE, 2004.

[2] Clack, Christopher D., Vikram A. Bakshi, and Lee Braine. "Smart contract templates: foundations, design landscape and research directions." arXiv preprint arXiv:1608.00771 (2016).

[3] Hazard, James, and Helena Haapio. "Wise contracts: smart contracts that work for people and machines." Trends and communities of legal informatics. Proceedings of the 20th international legal informatics symposium IRIS. 2017.

[4] Raskin, Max. "The law and legality of smart contracts." (2016).

[5] Shehata, Ibrahim. "Smart Contracts & International Arbitration." Available at SSRN 3290026 (2018).

[6] Subramanian, Hemang. "Decentralized blockchain-based electronic marketplaces." *Commun. ACM* 61.1 (2018): 78-84.

[7] Legalese. “Legalese.” Legalese, https://legalese.com/.

[8] Love, Nathaniel, and Michael Genesereth. "Computational law." Proceedings of the 10th international conference on Artificial intelligence and law. ACM, 2005.

[9] Hazard, J., and T. Hardjono. "CommonAccord: Towards a Foundation for Smart Contracts in Future Blockchains." Blockchains and the Web: W3C Workshop on Distributed Ledgers on the Web. 2016.

[10] Griffith, Verma. “Designs for the L4 Contract Programming Language” Legalese Presentation, https://www.bokconsulting.com.au/wp-content/uploads/2016/09/l4.pdf

[11] Luu, Loi, et al. "Making smart contracts smarter." Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. ACM, 2016.

[12] Bhargavan, Karthikeyan, et al. "Formal verification of smart contracts: Short paper." Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security. ACM, 2016.

[13] “Maritime Cargo Turnover - 6 Months 2019.” Port of Antwerp, https://www.portofantwerp.com/en/publications/statistics/maritime-cargo-turnover-6-months-2019

Annex I: D2.9 Blockchain Specifications

The D2.9 document contained work that fully envelops what is expected of a blockchain specification. Specifically, it analyses the needs of the LLs and provides some key indicators as to what the tangible and intangible requirements of the blockchain solution are. After deliberation with the consortium with regards to the initial mention of Hyperledger Fabric in the GA, we concluded that we should instead move forward with the Tendermint Consensus engine as it fit to the needs of the PI much better than the conventional Hyperledger Fabric solution. In summary, the Tendermint engine enables us to create a purpose-built blockchain in comparison to a readily available one such as HF as it opens up greater freedom and enables us to curate the produced code towards the actual needs of the LLs. Comparison and market investigation also led to Tendermint prevailing over HF in certain areas such as transactional throughput and network dispersion. These conclusions are outputs of the D2.9 v1 deliverable and can be observed there.

Annex II: Blockchain Source Code

As observed in the code below, several updates have been made with regards to how functions are defined and actuated. The updated source code encompasses all types of transactions required by the dynamic contract creation specified in this document.

```
const createABCIServer = require("abci");
const PORT = process.env.TENDERMINT_PORT || 26658;

// turn on debug logging
require("debug").enable("abci*");

const state = {};

const handlers = {
  info() {
    return {
      data: "ICONET Blockchain - PI SLA Contract Former",
      version: "0.8.0",
      lastBlockHeight: 0,
      lastBlockAppHash: Buffer.alloc(0)
    };
  },
  query({ path }) {
    return { code: 0, log: JSON.stringify(state[path]) };
  },
```

```
checkTx({ tx }) {
  const { valid, type, vars } = extractVariables(tx);

  if (!valid) {
    return {
      code: 1,
      log:
        "tx type " +
        type +
        ": tx failure, payload length incorrect for specified transaction type"
    };
  } else if (type === 1 && state[vars[0]] !== undefined) {
    return {
      code: 1,
      log:
        "tx type " +
        type +
        ": tx failure, SLA already exists for specified ID " +
        vars[0]
    };
  } else if (type !== 1 && state[vars[0]] === undefined) {
    return {
      code: 1,
      log:
        "tx type " +
        type +
        ": tx failure, no SLA exists for specified ID " +
        vars[0]
    };
  } else if (type !== 1 && state[vars[0]].finished) {
    return {
      code: 1,
      log:
        "tx type " +
        type +
        ": tx failure, attempted to set new measurement whilst packet has arrived at final
destination"
    };
  }

  return { code: 0, log: "tx succeeded" };
},
```

```
deliverTx({ tx }) {
  const { type, vars, valid } = extractVariables(tx);

  if (!valid) {
    return {
      code: 1,
      log:
        "tx type " +
        type +
        ": tx failure, payload length incorrect for specified transaction type"
    };
  } else if (type === 1 && state[vars[0]] !== undefined) {
    return {
      code: 1,
      log:
        "tx type " +
        type +
        ": tx failure, SLA already exists for specified ID " +
        vars[0]
    };
  } else if (type !== 1 && state[vars[0]] === undefined) {
    return {
      code: 1,
      log:
        "tx type " +
        type +
        ": tx failure, no SLA exists for specified ID " +
        vars[0]
    };
  } else if (type !== 1 && state[vars[0]].finished) {
    return {
      code: 1,
      log:
        "tx type " +
        type +
        ": tx failure, attempted to set new measurement whilst packet has arrived at final
destination"
    };
  }

  switch (type) {
    // 1 -> New Entry
    // ID, lowHumidity / lowTemperature, highHumidity / highTemperature / orientation,
typeOfRestriction, metadata
    case 1: {
      // Since orientation is defined by a single numerical value, we can assign the correct
orientation to the "low" object property for the comparisons that follow
      state[vars[0]] = {
        low: vars[1] || vars[2],
        high: vars[2],
        metadata: vars[3],
        events: [],
        broken: false,
        finished: false
      };

      break;
    }
    // 2 -> Update Entry
    // ID, measurement
    case 2: {
      if (state[vars[0]].low > vars[1] || state[vars[0]].high < vars[1]) {
        state[vars[0]].broken = true;
      }

      break;
    }
    // 3 -> Event Storage
    // ID, eventId
    case 3: {
      state[vars[0]].events.push(vars[1]);

      break;
    }
    // 4 -> Proof of Delivery
    // ID, eventId
    case 4: {
      state[vars[0]].events.push(vars[1]);
      state[vars[0]].finished = true;

      break;
    }
  }

  return { code: 0, log: "tx succeeded" };
}
};
```

```
function extractVariables(tx) {
  const vars = [];

  switch (tx[0]) {
    // 1 -> New Entry
    // ID, lowHumidity / lowTemperature, highHumidity / highTemperature / orientation, metadata
    case 1: {
      if (tx.length <= 13) return { valid: false };

      vars.push(tx.readUInt32BE(1));
      vars.push(tx.readUInt32BE(5));
      vars.push(tx.readUInt32BE(9));
      vars.push(tx.slice(13).toString());
      break;
    }
    // 2 -> Update Entry
    // ID, measurement
    case 2: {
      if (tx.length !== 9) return { valid: false };

      vars.push(tx.readUInt32BE(1));
      vars.push(tx.readUInt32BE(5));
      break;
    }
    // 3 -> Event Storage
    // ID, eventId
    // 4 -> Proof of Delivery
    // ID, eventId
    case 3:
    case 4: {
      if (tx.length <= 5) return { valid: false };

      vars.push(tx.readUInt32BE(1));
      vars.push(tx.slice(5).toString());
      break;
    }
  }

  return { type: tx[0], vars, valid: true };
}

createABCIServer(handlers).listen(PORT, () => {
  console.log(`listening on port ${PORT}`);
});
```

Annex III: RESTful Server Source Code

```
const crypto = require("crypto");
const fastify = require("fastify")();
const instance = require("axios").create({
  baseURL: `http://localhost:${process.env.TENDERMINT_PORT || 26657}`,
  timeout: 1000
});
const PORT = process.env.FASTIFY_PORT || 7777;

function getLimits({
  requiredOrientation,
  requiredHumidityRange,
  requiredTemperatureRange
}) {
  if (requiredOrientation !== undefined) {
    return [requiredOrientation, requiredOrientation];
  } else return requiredHumidityRange || requiredTemperatureRange;
}

function hash(...args) {
  const sha256 = crypto.createHash("sha256");
  args.forEach(arg => sha256.update(arg));
  return sha256.digest("hex");
}

function encodeToBytes(...args) {
  let encoded = `0x0${args.shift()}`;
  args.forEach(arg => {
    let toAdd;

    if (isNaN(arg)) {
      toAdd = toHex(arg);
    } else {
      toAdd = toUint32(arg);
    }

    if (toAdd.length % 2 !== 0) toAdd = "0" + toAdd;

    encoded += toAdd;
  });
  return encoded;
}
```

```
function toUint32(num) {
  return ("0000000" + num.toString(16).toUpperCase()).slice(-8);
}

function toHex(s) {
  s = unescape(encodeURIComponent(s));
  let h = "";
  for (let i = 0; i < s.length; i++) {
    h += s.charCodeAt(i).toString(16);
  }

  return h;
}

function convert(str) {
  if (isNaN(str))
    str = str
      .split("")
      .reduce((acc, val, i) => acc + val.charCodeAt(0) * (i + 1), 0);
  return str % 4294967295;
}

// Enable if transaction indexing is enabled
//
// fastify.get("/tx/:hash", async r => {
//   const { data } = await instance.get("/tx", {
//     params: {
//       hash: `"${r.params.hash.toLowerCase()}"`
//     }
//   });

//   if (data.error) {
//     return {
//       message: "Failed to retrieve data from blockchain transaction hash",
//       error: data.error
//     };
//   } else {
//     return {
//       message: "Successfully retrieved blockchain transaction from hash",
//       contract: data.result
//     };
//   }
// });
```

```
fastify.post("/contract", async r => {
  // We expect the r.body payload to comply with the ICONET Domain Model Order
  const { origin, destination, containers, products, status } = r.body;

  const hashes = [];

  for (let i = 0; i < containers.length; i++) {
    hashes.push(
      await instance.get("/broadcast_tx_commit", {
        params: {
          tx: encodeToBytes(
            1,
            convert(containers[i].identifier),
            ...getLimits(
              products.find(
                ({ piContainer }) => piContainer === containers[i].identifier
              )
            ),
            hash(origin, destination, status)
          )
        }
      })
    );
  }

  if (hashes.some(({ data: { error } }) => error)) {
    return {
      message: "Failed to create blockchain contracts",
      errors: hashes.map(({ data: { error } }) => error)
    };
  }

  return {
    message: "Successfully created blockchain contracts",
    hashes: hashes.map(
      ({
        data: {
          result: { hash }
        }
      }) => hash
    )
  };
});

fastify.get("/contract/:id", async r => {
  const { data } = await instance.get("/abci_query", {
    params: {
      path: `"${convert(r.params.id)}"`
    }
  });

  if (data.result)
    return {
      message: "Successfully retrieved contract blockchain state",
      contract: JSON.parse(data.result.response.log || "{}")
    };
  else
    return {
      message: "Failed to retrieve contract blockchain state",
      error: data.error
    };
});
```

```
fastify.patch("/contract/:id/metric", async r => {
  const { data } = await instance.get("/broadcast_tx_sync", {
    params: {
      tx: encodeToBytes(2, parseInt(r.params.id), r.body.metric)
    }
  });
  return {
    message: "Successfully submitted metric to the blockchain contract",
    hash: data.result ? data.result.hash : data
  };
});

fastify.patch("/contract/:id/event", async r => {
  const { data } = await instance.get("/broadcast_tx_sync", {
    params: {
      tx: encodeToBytes(
        r.body.event.final ? 4 : 3,
        parseInt(r.params.id),
        hash(JSON.stringify(r.body.event))
      )
    }
  });
  return {
    message: "Successfully submitted event to the blockchain contract",
    hash: data.result ? data.result.hash : data
  };
});

fastify
  .listen(PORT)
  .then(() => console.log(`Fastify is listening on port ${PORT}`));

module.exports = fastify;
```

Annex IV: Full Test Suite

```
const tap = require("tap");

const crypto = require("crypto");

const fastify = require("../server");

const generateId = () =>
  Math.random()
    .toString(36)
    .slice(-5);

const wait = ms => new Promise(resolve => setTimeout(resolve, ms));

function hash(...args) {
  const sha256 = crypto.createHash("sha256");
  args.forEach(arg => sha256.update(arg));
  return sha256.digest("hex");
}

function waitFor(id, member) {
  return wait(1500).then(
    () =>
      new Promise((resolve, reject) => {
        fastify.inject(
          {
            method: "GET",
            url: `/contract/${id}`
          },
          (err, response) => {
            if (err) reject(err);

            resolve(JSON.parse(response.body)[member]);
          }
        );
      })
  );
}
```

```
const identifier = generateId();

const exampleContainer = {
  origin: "Greece/Athens",
  destination: "Krakow/Poland",
  containers: [{ identifier }],
  products: [{ piContainer: identifier, requiredTemperatureRange: [10, 20] }],
  status: "in-transit"
};

tap.tearDown(() => fastify.close());

tap.test("POST `/contract` route single", t => {
  t.plan(4);

  fastify.inject(
    {
      method: "POST",
      url: `/contract`,
      body: exampleContainer
    },
    (err, response) => {
      t.error(err);
      const {
        message,
        hashes: [hash]
      } = JSON.parse(response.body);
      t.strictEqual(message, "Successfully created blockchain contracts");
      t.assert(typeof hash === "string");
      t.assert(hash.length === 64);
    }
  );
});
```

```
tap.test("POST `/contract` route multi", t => {
  t.plan(6);

  const id1 = generateId();
  const id2 = generateId();

  fastify.inject(
    {
      method: "POST",
      url: `/contract`,
      body: {
        origin: "Greece/Athens",
        destination: "Krakow/Poland",
        containers: [{ identifier: id1 }, { identifier: id2 }],
        products: [
          { piContainer: id2, requiredTemperatureRange: [10, 20] },
          { piContainer: id1, requiredHumidityRange: [5, 10] }
        ],
        status: "in-transit"
      }
    },
    (err, response) => {
      t.error(err);
      const {
        message,
        hashes: [hash1, hash2]
      } = JSON.parse(response.body);
      t.strictEqual(message, "Successfully created blockchain contracts");
      t.assert(typeof hash1 === "string");
      t.assert(hash1.length === 64);
      t.assert(typeof hash2 === "string");
      t.assert(hash2.length === 64);
    }
  );
});
```

```
tap.test("GET `/contract/:id` route", t => {
  t.plan(9);

  fastify.inject(
    {
      method: "GET",
      url: `/contract/${identifier}`
    },
    (err, response) => {
      t.error(err);

      const {
        message,
        contract: { low, high, metadata, events, broken, finished }
      } = JSON.parse(response.body);

      t.strictEqual(
        message,
        "Successfully retrieved contract blockchain state"
      );

      t.strictEqual(
        low,
        exampleContainer.products[0].requiredTemperatureRange[0]
      );

      t.strictEqual(
        high,
        exampleContainer.products[0].requiredTemperatureRange[1]
      );

      t.assert(Array.isArray(events));

      t.assert(events.length === 0);

      t.assert(!broken);

      t.assert(!finished);

      t.assert(
        metadata,
        hash(
          exampleContainer.origin,
          exampleContainer.destination,
          exampleContainer.status
        )
      );
    }
  );
});
```

```
tap.test("PATCH `/contract/:id/metric` route", t => {
  t.plan(4);

  fastify.inject(
    {
      method: "PATCH",
      url: `/contract/${identifier}/metric`,
      body: {
        metric: 15
      }
    },
    (err, response) => {
      t.error(err);
      const { message, hash } = JSON.parse(response.body);
      t.strictEqual(
        message,
        "Successfully submitted metric to the blockchain contract"
      );
      t.assert(typeof hash === "string");
      t.assert(hash.length === 64);
    }
  );
});

tap.test("PATCH `/contract/:id/metric` route (breaks contract)", t => {
  t.plan(2);

  fastify.inject(
    {
      method: "PATCH",
      url: `/contract/${identifier}/metric`,
      body: {
        metric: 25
      }
    },
    (err, response) => {
      t.error(err);

      waitFor(identifier, "broken")
        .then(val => {
          t.assert(val);
        })
        .catch(err => {
          t.error(err);
        });
    }
  );
});
```

```
tap.test("PATCH `/contract/:id/event` route", t => {
  t.plan(4);

  fastify.inject(
    {
      method: "PATCH",
      url: `/contract/${identifier}/event`,
      body: {
        event: {
          final: false,
          data: "delayed",
          random: "attribute"
        }
      }
    },
    (err, response) => {
      t.error(err);
      const { message, hash } = JSON.parse(response.body);
      t.strictEqual(
        message,
        "Successfully submitted event to the blockchain contract"
      );
      t.assert(typeof hash === "string");
      t.assert(hash.length === 64);
    }
  );
});

tap.test("PATCH `/contract/:id/event` route (final event)", t => {
  t.plan(2);

  fastify.inject(
    {
      method: "PATCH",
      url: `/contract/${identifier}/event`,
      body: {
        event: {
          final: true,
          some: "state"
        }
      }
    },
    (err, response) => {
      t.error(err);

      waitFor(identifier, "finished")
        .then(val => {
          t.assert(val);
        })
        .catch(err => {
          t.error(err);
        });
    }
  );
});
```

Annex V: Benchmark Suite for LL1

```
"use strict";

const axios = require("axios");

const autocannon = require("autocannon");

async function bench() {
  await axios.post("http://localhost:7777/contract", {
    origin: "Greece/Athens",
    destination: "Krakow/Poland",
    containers: [{ identifier: "cont-5" }],
    products: [{ piContainer: "cont-5", requiredTemperatureRange: [10, 20] }],
    status: "in-transit"
  });
  const LL1 = autocannon(
    {
      method: "PATCH",
      url: "http://localhost:7777/contract/1658/metric",
      headers: {
        "Content-Type": "application/json; charset=utf-8"
      },
      body: JSON.stringify({ metric: 15 })
    },
    console.log
  );

  autocannon.track(LL1);
}

bench();
```

Annex VI: Benchmark Suite for LL2

```
"use strict";

const autocannon = require("autocannon");

async function bench() {
  const LL2 = autocannon(
    {
      url: "http://localhost:7777/contract",
      method: "POST",
      headers: {
        "Content-Type": "application/json; charset=utf-8"
      },
      body: JSON.stringify({
        origin: "Greece/Athens",
        destination: "Krakow/Poland",
        containers: [{ identifier: "[<id>]" }],
        products: [{ piContainer: "[<id>]", requiredHumidityRange: [10, 20] }],
        status: "in-transit"
      }),
      idReplacement: true
    },
    console.log
  );
  autocannon.track(LL2);
}

bench();
```

Annex VII: Benchmark Suite for LL3

```
"use strict";

const autocannon = require("autocannon");

function convert(str) {
  if (isNaN(str))
    str = str
      .split("")
      .reduce((acc, val, i) => acc + val.charCodeAt(0) * (i + 1), 0);
  return str % 4294967295;
}

async function bench() {
  let i = 0;
  let j = 0;
  const LL3 = autocannon(
    {
      url: "http://localhost:7777",
      requests: [
        {
          method: "POST",
          path: "/contract",
          headers: {
            "Content-Type": "application/json; charset=utf-8"
          },
          setupRequest: req => {
            req.body = JSON.stringify({
              origin: "Greece/Athens",
              destination: "Krakow/Poland",
              containers: [{ identifier: "cont-" + i }],
              products: [
                {
                  piContainer: "cont-" + i,
                  requiredHumidityRange: [10, 20]
                }
              ],
              status: "in-transit"
            });
            return req;
          }
        },
```

```
        {
          method: "PATCH",
          body: JSON.stringify({
            metric: 21
          }),
          setupRequest: req => {
            req.path = `/contract/${convert("cont-" + j)}/metric`;
            return req;
          }
        }
      ]
    },
    console.log
  );

  autocannon.track(LL3);
}

bench();
```

Annex IIX: Benchmark Suite for LL4

```
"use strict";

const axios = require("axios");

const autocannon = require("autocannon");

async function bench() {
  await axios.post("http://localhost:7777/contract", {
    origin: "Greece/Athens",
    destination: "Krakow/Poland",
    containers: [{ identifier: "cont-5" }],
    products: [{ piContainer: "cont-5", requiredTemperatureRange: [10, 20] }],
    status: "in-transit"
  });
  const LL4 = autocannon(
    {
      url: "http://localhost:7777",
      headers: {
        "Content-Type": "application/json; charset=utf-8"
      },
      method: "PATCH",
      requests: [
        {
          path: "/contract/1658/metric",
          body: JSON.stringify({ metric: 15 })
        },
        {
          path: "/contract/1658/event",
          body: JSON.stringify({
            event: {
              final: false,
              type: "TransportInitiated",
              value: 1
            }
          })
        }
      ]
    },
    console.log
  );
  autocannon.track(LL4);
}

bench();
```