TECHNICAL PAPER

Unibright – the unified framework for blockchain based business integration

unibright.io

team@unibright.io

UNIBRIGHT TECHNICAL DETAILS

March 3rd 2018

Stefan Schmidt, Unibright CTO (stefan@unibright.io)

Marten Jung, Unibright CEO (marten@unibright.io)

This document describes the technical details of the underlying concept of the Unibright Framework. It covers the components, their technical setup, software architectural principals and also presents a walkthrough with code examples.

This document should be seen as an add-on to the Unibright whitepaper (https://unibright.io/files/Unibright_Whitepaper.pdf)

Table of Content

Unibright Framework Components

The Unibright Framework consists of 4 components:

Workflow Designer (WD) (Web-Application): Visually defining business integration workflows, based on a use-case specific template

Contract Lifecycle Manager (CLM) (Web-Application): Loading previously defined workflows and generating smart contract code, smart adapters and smart queries automatically

Connector (UBC) (Background Task with minimal visual frontend): Connecting previously generated smart contracts to other systems/smart contracts by previously generated smart adapters

Explorer (EX) (Web-Application): Monitoring existing processes by displaying data/information from generated smart contracts and connecting systems.

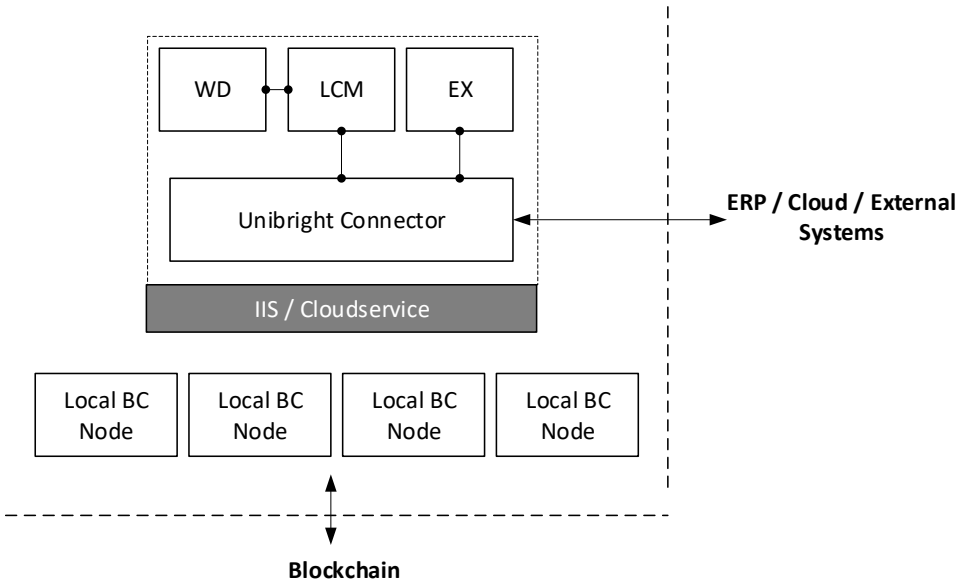


Fig: Unibright Framework Components

The Unibright Workflow Designer (WD), Contract Lifecycle Manager (CLM) and Explorer (EX) are Web applications, build with frontend Libraries like React and Rapid. They are developed as Microsoft asp.net Applications and can be either hosted locally or in the cloud (see “System setups”).

The Unibright Connector (UBC) is based on a Microsoft .NET class library, it connects to blockchain nodes and external systems.

CLM and EX use UBC to publish and query the blockchains and all connected systems.

System setups

The Unibright Framework can be run and maintained in different system setups. The decision for a specific setup is driven by the customer's needs in terms of access control, network integration, maintenance and availability.

"On Premise" (local IIS in company's IT setup)

a. WD, CLM, and EX are hosted on a local IIS
b. UBC is run by the local IIS
c. UBC connect to local nodes of blockchains to be integrated
d. UBC connects to other systems (ERP) by channels available in the local network

2. Cloud based SAAS
    a. WD, CLM and EX are hosted in a pre-defined MS Azure cloud setup
    b. UBC is run as an MS Azure Classic Cloudservice
    c. UBC connects to blockchain nodes of blockchains to be integrated via the adapters available on MS azure
    d. UBC connects to other systems (e.g. ERP) by channels accessible from MS Azure

3. Cloud based PAAS

    (Setup like 2, but pre-setup in a Virtual Machine, possibly run and maintained by a System Integration Partner.

Software Architectural Principles

The central part of the Unibright ecosystem is the Unibright Contract Interface (UCI): The UCI defines the main structure, state variables, mappings and methods which every generated Smart Contract automatically implements, thus marking a smart contract Unibright conformant. It is the irremovable guarantee of recognizing smart contracts as part of the Unibright ecosystem, ensuring that Unibright conformant smart contracts can be found, called, maintained and connected.

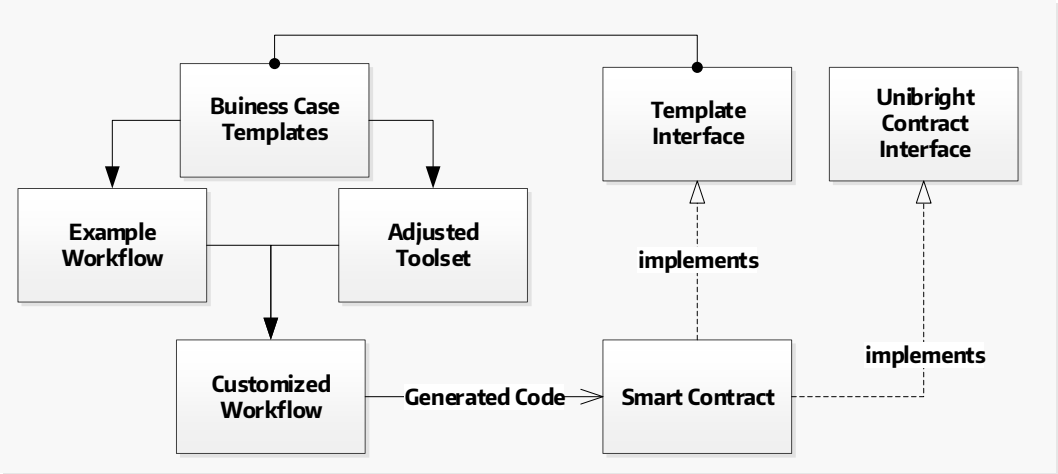


Fig: Interfaces and Hierarchy

Content-wise, the UCI offers the fundament to integrate smart contracts into different blockchains and system landscapes. Operators of the Unibright ecosystem can visually define workflows and choose from a set of Templates.

Templates pre-define typical business workflows on a high level of abstraction. By choosing a certain template, the operator is automatically given a suitable subset of all available workflow tool set items and an initial example workflow which can be customized to the needs of a special use case. Each template brings its own purpose-built interface, which the generated smart contract will implement in addition to the basic Unibright Contract Interface.

System boundaries and data exchange

The Workflow Designer knows the Unibright Templates and presents a basic workflow based on one of these Templates (combination of Templates will also be available). The integration workflow can be customized. The current state of work can be stored locally or in the cloud as a JSON representation. WD has no connection to any blockchain or backend system.

The Contract Lifecycle Manager (CLM) loads a formerly saved JSON workflow representation, being able to rebuild the graphical representation of the workflow. CLM defines a specific blockchain target, the type and version of connected systems and the Channels the communication takes place on (see Adapter section for Details). CLM then creates smart contract code, which can be saved be saved locally as well. After publishing code to the specific blockchain, smart adapter configurations and smart query sets are created (XML), they can be stored locally or in the cloud, together with the Workflow Representation as one package file

The Explorer opens created query sets (XML) to show the information and transaction flow, making use of the generated smart contract code (targeting Blockchain) and all smart adapters (targeting connected systems). Query sets can be adjusted and saved as XML files to be used later. Graphical representations can be saved as PDF or PNG.

The Connector uses the provided XML based adapter configurations to establish the desired connections to all participating systems, making use of the predefined structure of the Unibright Contract Interface.

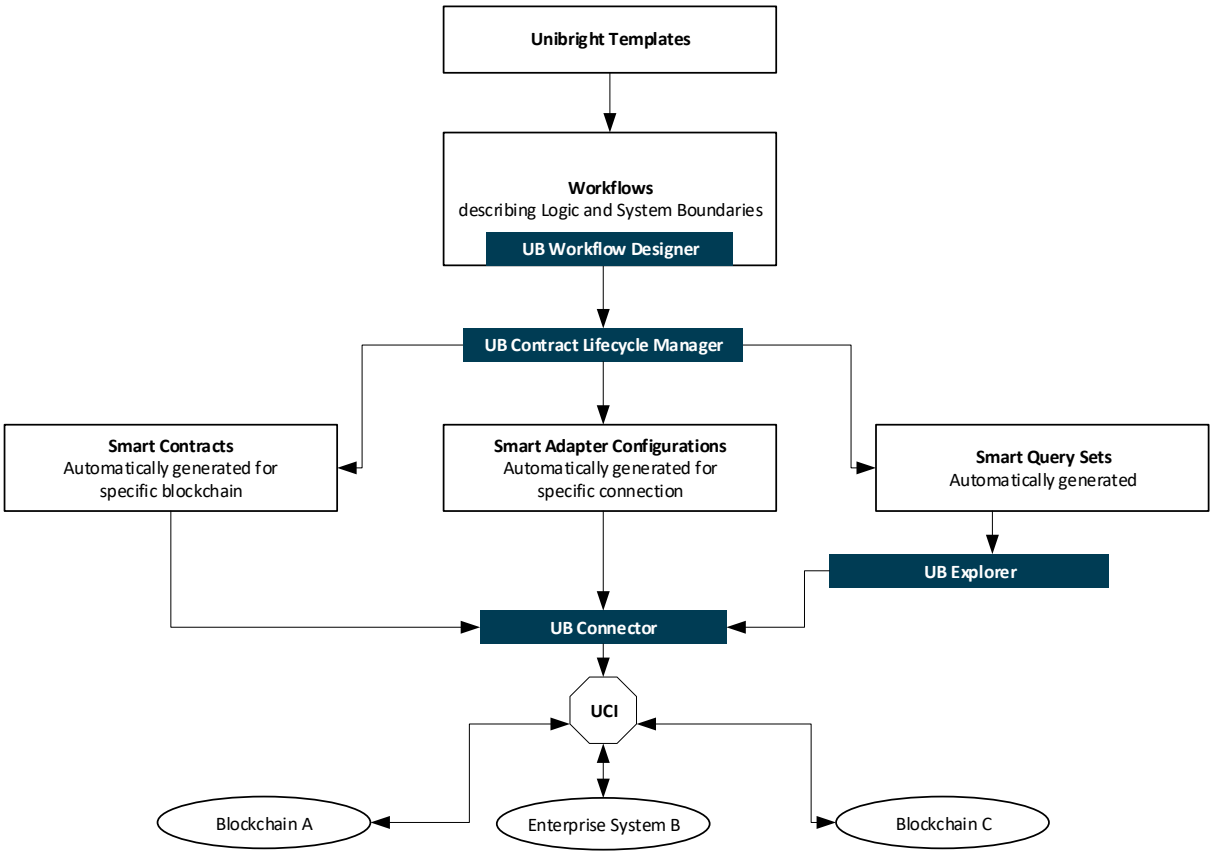


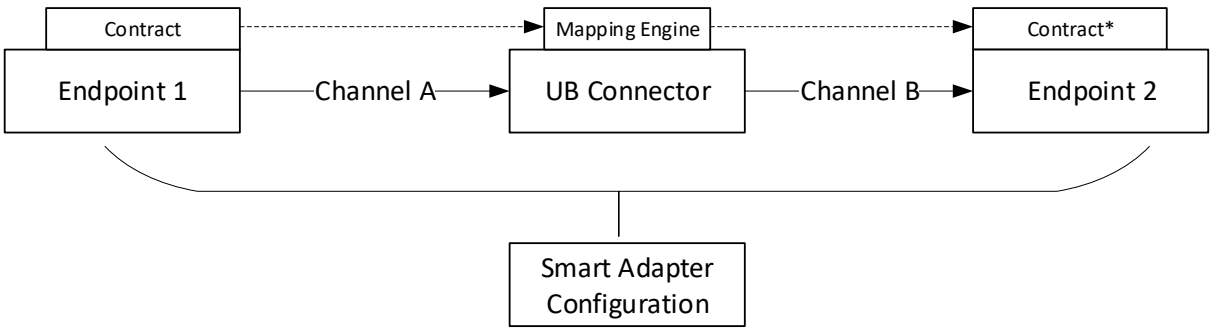
Fig: Objects and system components in the Unibright Framework

Unibright connector and smart adapters

The Unibright Connector (UBC) is based on the coding of the cloud-based process integration platform PIP, owned by SPO Consulting GmbH (Unibright is a management spin-off) and 100% licenced to Unibright without any restrictions. PIP is live since 2011, serves productive customers from Banking to productive sector and moved to Microsoft Azure in 2015, as Software-as-a-Service.

UBC uses endpoints, channels, contract interfaces and mappings:

- An endpoint is the source or destination of an integration process
- A channel is the technical part of a connection between an endpoint and the connector. Unibright offers
    - SOAP based webservices
    - REST
    - FTP reading (also polling) and writing
    - RFC
    - SAP IDOC
    - Reading/sending data from/via email attachments
    - Reading and writing to Databases
    - Accessing local blockchain nodes (individual channel per blockchain implementation)
- A contract interface is an object representation for a specific target system (represented by an endpoint), for example ORDERS01 IDOC in SAP
- A mapping is an xml-based description on mapping and transforming one contract into another, taking into account object specific constraints on datatypes, data lengths, formatting and unit systems



A smart adapter is a plugin code unit, which is configured by an XML configuration file, telling the adapter which contract should be send or received on which channel by which mapping. The configuration files are automatically generated from the designed Templates.

Walkthrough and Code Examples

This section describes a walkthrough through the process and gives some code examples. The underlying template has been simplified for explaining the concept rather than explaining code details.

Example Workflow: Request for Quotation-Template

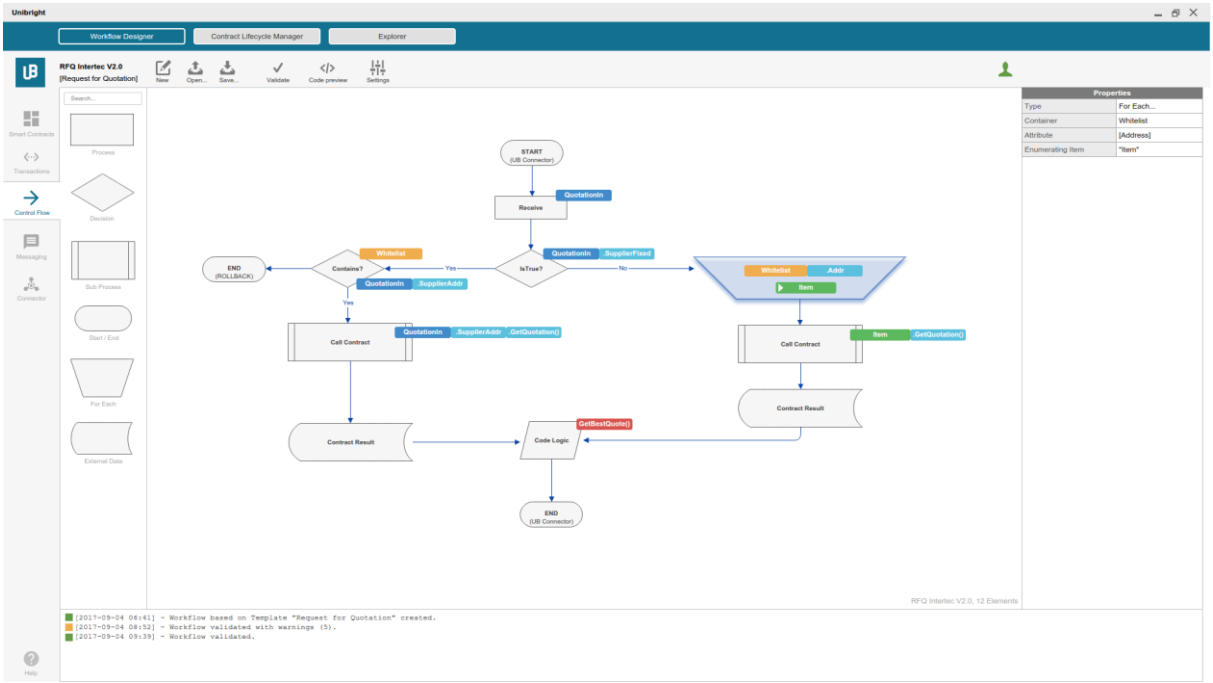


Fig: Example Workflow

The Example given shows the workflow for “Request for Quotation”.

The workflow starts by receiving a “QuotationIN” Object from the UBC. This Object is related to the Template “Request for Quotation” and holds the specific parameters needed to empower the use case. It inherits from “Unibright Contract”, the base class that provides basic attributes like a “whitelist”, the contract Address and static Methods like “Receive” to start the process.

```
6 references
58  public class QuotationIN : UnibrightContract
59  {
60      private bool supplierFixed;
61      private string supplierAddress;
62      private string material;
63      private string unit;
64      private decimal amount;
65      private DateTime latestDeliveryDate;
66
        2 references
67      public bool SupplierFixed { get => supplierFixed; set => supplierFixed = value; }
        5 references
68      public string SupplierAddress { get => supplierAddress; set => supplierAddress = value; }
        2 references
69      public string Material { get => material; set => material = value; }
        2 references
70      public string Unit { get => unit; set => unit = value; }
        2 references
71      public decimal Amount { get => amount; set => amount = value; }
        2 references
72      public DateTime LatestDeliveryDate { get => latestDeliveryDate; set => latestDeliveryDate = value; }
73  } ↑ class QuotationIN
74
```

Fig: Example Code for Template related QuotationIN-object

In the next step of the workflow it is checked if the supplier for this case is fixed. If so (left branch), it is checked if the whitelist contains the current's supplier address and the smart contract holding the supplier logic is called. The Contract result is then evaluated by custom code (given by the template) and handed to the connector as well. The right branch (supplier is not fixed), iterates through a list of given supplier smart contract addresses, receiving their quotation (asynchronous) and passing it the custom code evaluation.

Workflow Designer representation

```
{
  "activities": [
    {
      "id": "YcJsrUtEih",
      "hover": {
        "opacity": 0.2
      },
      "cursor": "pointer",
      "content": {
        "align": "center middle",
        "color": "#2e2e2e"
      },
      "selectable": true,
      "serializable": true,
      "enable": true,
      "type": "Receive",
      "UBcontract": "QuotationIN",
      "path": "",
      "autoSize": true,
      "visual": null,
      "x": 540,
      "y": 630,
      "minWidth": 20,
      "minHeight": 20,
      "width": 120,
      "height": 120,
      "editable": {
        "connect": true,
        "tools": [],
        "drag": {
          "snap": {
            "size": 10,
            "angle": 10
          }
        },
        "remove": true
```

Fig: JSON-Representation of workflow item

The representation of the workflow is a JSON-serialization. It consists of graphical representations (position, colours, connections) and of template related attributes (like "type" or "UBContract"). It holds no code logic and can be saved locally.

Templates, Workflow, Elements, Activities

When loaded in the Contract Lifecycle Manager, the JSON representation is automatically mapped into an XML-File, representing a serialized version of a .NET object structure.

This object structure consists of

- “Workflow” class, consisting of Elements and edges
- “WorkflowElement” class, representing a single workflow element consisting of an Activity, metadata (ID, caption) and pointers to the predecessor(s) and successor(s) in the workflow
- “Activity” class, being the base class for a specific activity in a workflow element, e.g. “Is-True”-Activity only evaluating a single Boolean expression or “Call Contract”-Activity being able to call a smart contract function
- “Template” class, holding a basic workflow and a list of allowed Activities for any template based specific workflow

The classes are built as C# generics, allowing a solid basic architecture.

```
9   namespace Unibright.Workflows
10  {
11      #region Workflows
12
            4 references
13      public class Workflow
14      {
15          private List<WorkflowElement> elements = new List<WorkflowElement>();
16          private List<Tuple<WorkflowElement, WorkflowElement, EdgeType>> orientedEdges = new List<Tuple<WorkflowElement, WorkflowElement, Edg
17
                5 references
18          internal void AddWorkflowElement<T>(T activity) where T : Activity
19          {
20              var we = new WorkflowElement(activity, "");
21              this.elements.Add(we);
22          } ↑ void AddWorkflowElement(T)
23
                4 references
24          private WorkflowElement GetWorkflowElementByActivity(Activity a)
25          {
26              var element = elements.FirstOrDefault(e => e.Activity == a);
27              if (element == null)
28              {
29                  element = new WorkflowElement(a, "");
30                  elements.Add(element);
31              }
32              return element;
33          } ↑ WorkflowElement GetWorkflowElementByActivity(Activity)
34
                2 references
35          internal void AddTrueFalseEdge<T>(AT_Conditioned<T> istrue_activity, Activity result_activity, bool isTrue) where T : UnibrightCont
36          {
37              var istrue_element = GetWorkflowElementByActivity(istrue_activity);
38              var result_element = GetWorkflowElementByActivity(result_activity);
39
40              istrue_element.Successors.Add(result_element);
41              result_element.Predecessors.Add(istrue_element);
42
43              var edgetype = EdgeType.False;
44              if (isTrue)
45                  edgetype = EdgeType.False;
46
47              orientedEdges.Add(new Tuple<WorkflowElement, WorkflowElement, EdgeType>(istrue_element, result_element, edgetype));
48          } ↑ void AddTrueFalseEdge(AT_Conditioned<T>, Activity, bool)
49
                2 references
50          internal void AddEdge(Activity source_activity, Activity result_activity)
51          {
52              var source_element = GetWorkflowElementByActivity(source_activity);
53              var result_element = GetWorkflowElementByActivity(result_activity);
54
55              source_element.Successors.Add(result_element);
56              result_element.Predecessors.Add(source_element);
57
58              orientedEdges.Add(new Tuple<WorkflowElement, WorkflowElement, EdgeType>(source_element, result_element, EdgeType.Directed));
59          } ↑ void AddEdge(Activity, Activity)
```

Fig: Excerpt from C#-Class representing a Workflow

The XML-object structure of a workflow can be deserialized to an object again. The same object can be built by a factory in C#:

```
209         try
210         {
211             Workflow rfqWorkflow = new Workflow();
212
213             var a_start = new AT_Start();
214             var a_receive = new AT_Receive<QuotationIN>();
215             var quotationIN = a_receive.Receive();
216
217             var a_istrue = new AT_IsTrue<QuotationIN>(p => p.SupplierFixed);
218             var a_contains = new AT_Contains<string, QuotationIN>
219                 (quotationIN.Whitelist.Select(wh => wh.ContractAddress).ToList(), p => p.SupplierAddress);
220             var a_rollback = new AT_Rollback();
221             var a_callContract = new AT_CallContract<QuotationIN, QuotationSupplier, QuotationOUT>
222                 (p => UnibrightContract.Receive<QuotationSupplier>(p.SupplierAddress), p => p.GetQuotation(quotationIN));
223             var a_codeLogic = new AT_CodeLogic<QuotationOUT>();
224
225             rfqWorkflow.AddWorkflowElement(a_start);
226             rfqWorkflow.AddWorkflowElement(a_receive);
227             rfqWorkflow.AddEdge(a_start, a_receive);
228
229             rfqWorkflow.AddWorkflowElement(a_istrue);
230             rfqWorkflow.AddEdge(a_receive, a_istrue);
231
232             rfqWorkflow.AddWorkflowElement(a_contains);
233             rfqWorkflow.AddTrueFalseEdge(a_istrue, a_contains, true);
234
235             rfqWorkflow.AddWorkflowElement(a_rollback);
236             rfqWorkflow.AddTrueFalseEdge(a_contains, a_rollback, true);
237
```

Fig: Example Code for a Factory building the RFQ-Workflow representation

Code-Generation

For code generation, a T4-File traverses the object structure and transforms it to platform specific code [https://msdn.microsoft.com/de-de/en-en/library/ee844259.aspx]

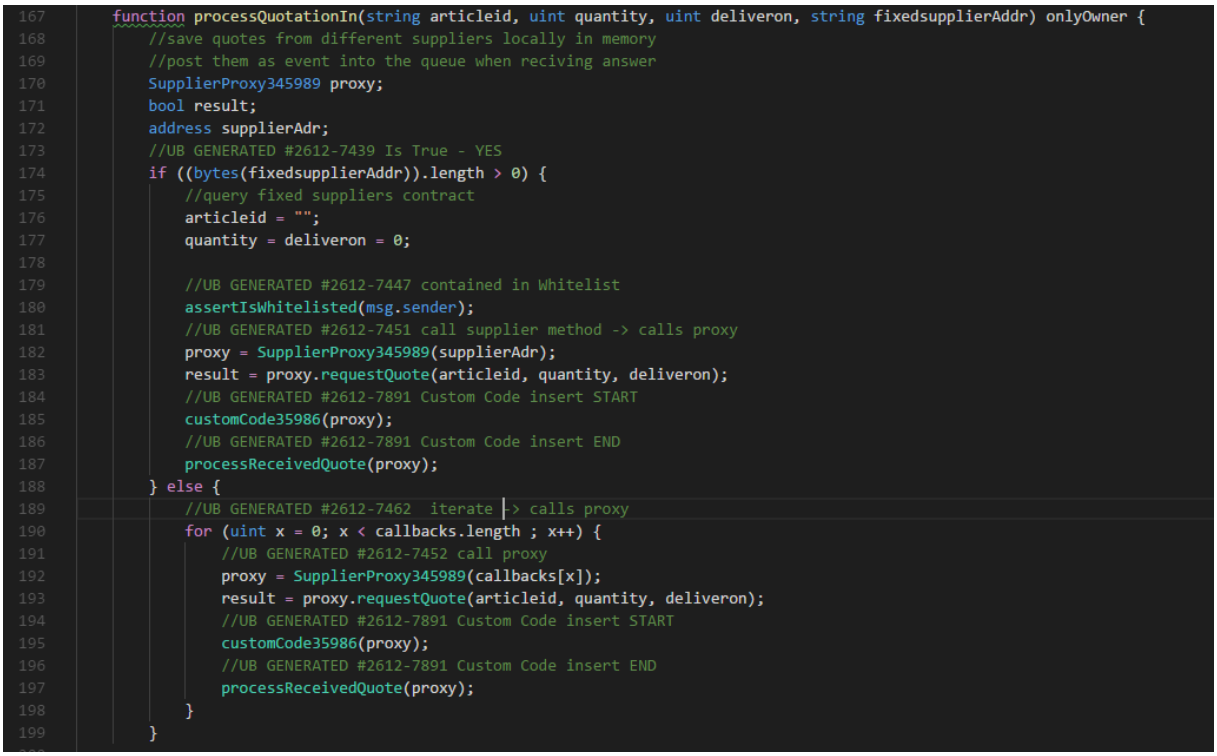
Each destination programming language needs its own T4-File, translating the workflow object structure into specific code.

The template specific main code is tested and audited for each of the available implementations. The workflow can be customized in terms of control structure, connecting existing systems and calling other smart contracts or oracles. These customizations are faced by the code generation files.

The same workflow structure can result in complete different implementations, e.g. .NET and Solidity.

```
// WORKFLOW GENERATION #2565-7433  Transaction
using (UnibrightTransaction ubt = UnibrightContract.StartTransaction())
{
    // WORKFLOW GENERATION #2565-7436  Receive
    QuotationIN quotationIN = UnibrightContract.Receive<QuotationIN>();
    string bestQuote_adr = "";

    // WORKFLOW GENERATION #2565-7439  IsTrue - YES
    if (quotationIN.SupplierFixed)
    {
        // WORKFLOW GENERATION #2565-7447  Contains - YES
        if (quotationIN.Whitelist.Select(w => w.ContractAddress).Contains(quotationIN.SupplierAddress))
        {
            // WORKFLOW GENERATION #2565-7451  Call Contract
            var supplier = UnibrightContract.Receive<QuotationSupplier>(quotationIN.SupplierAddress);
            var result = supplier.GetQuotation(quotationIN.Material, quotationIN.Unit, quotationIN.Amount, quotationIN.LatestDeliveryDate);

            // WORKFLOW GENERATION #2565-7891  CustomCode
            bestQuote_adr = CustomCode.GetBestQuote(result);

            // WORKFLOW GENERATION #2565-7434  Confirm
            ubt.Confirm();
        }
        // WORKFLOW GENERATION #2565-7447  Contains - YES
        else
        {
            // WORKFLOW GENERATION #2565-7435  Confirm
            ubt.Rollback();
        }
    }
    // WORKFLOW GENERATION #2565-7439  IsTrue - NO
    else
    {
        // WORKFLOW GENERATION #2565-7462  Foreach
        List<ContractResult<QuotationOUT>> results = new List<ContractResult<QuotationOUT>>();
        foreach (var item in quotationIN.Whitelist.Select(w => w.ContractAddress))
        {
            // WORKFLOW GENERATION #2565-7452  Call Contract
            var supplier = UnibrightContract.Receive<QuotationSupplier>(quotationIN.SupplierAddress);
            results.Add(supplier.GetQuotation(quotationIN.Material, quotationIN.Unit, quotationIN.Amount, quotationIN.LatestDeliveryDate));
        }
        // WORKFLOW GENERATION #2565-7891  CustomCode
        bestQuote_adr = CustomCode.GetBestQuote(results.ToArray());

        // WORKFLOW GENERATION #2565-7434  Confirm
        ubt.Confirm();
    }
}
```

Fig: Generated C# code out of RFQ-Workflow



```
function processQuotationIn(string articleid, uint quantity, uint deliveron, string fixedsupplierAddr) onlyOwner {
    //save quotes from different suppliers locally in memory
    //post them as event into the queue when reciving answer
    SupplierProxy345989 proxy;
    bool result;
    address supplierAdr;
    //UB GENERATED #2612-7439 Is True - YES
    if ((bytes(fixedsupplierAddr)).length > 0) {
        //query fixed suppliers contract
        articleid = "";
        quantity = deliveron = 0;

        //UB GENERATED #2612-7447 contained in Whitelist
        assertIsWhitelisted(msg.sender);
        //UB GENERATED #2612-7451 call supplier method -> calls proxy
        proxy = SupplierProxy345989(supplierAdr);
        result = proxy.requestQuote(articleid, quantity, deliveron);
        //UB GENERATED #2612-7891 Custom Code insert START
        customCode35986(proxy);
        //UB GENERATED #2612-7891 Custom Code insert END
        processReceivedQuote(proxy);
    } else {
        //UB GENERATED #2612-7462  iterate -> calls proxy
        for (uint x = 0; x < callbacks.length ; x++) {
            //UB GENERATED #2612-7452 call proxy
            proxy = SupplierProxy345989(callbacks[x]);
            result = proxy.requestQuote(articleid, quantity, deliveron);
            //UB GENERATED #2612-7891 Custom Code insert START
            customCode35986(proxy);
            //UB GENERATED #2612-7891 Custom Code insert END
            processReceivedQuote(proxy);
        }
    }
```

Fig: Excerpt of generated solidity code out of RFQ workflow

Publishing / Deploying

Smart contract code

The compiled smart contract code can be published via the Contract Lifecycle Manager (CLM). The CLM is connected to a local blockchain node via the Unibright Connector. For Ethereum, the web3.js-Library is used to deploy the smart contract.

```
125     public async Task<string> DeploySmartContract(string bytecode, string abi, params object[] p)
126     {
127
128         var mineResult = await web3.Miner.Start.SendRequestAsync(2).ConfigureAwait(false);
129
130         var receipt = await web3.Eth.DeployContract.SendRequestAndWaitForReceiptAsync
131             (abi, bytecode, _accountAddress, new HexBigInteger(900000), null, p).ConfigureAwait(false);
132
133         var mineResult2 = await web3.Miner.Stop.SendRequestAsync().ConfigureAwait(false);
134
135         var contractadr = receipt.ContractAddress;
136
137         return contractadr;
138
139     } ↑ Task<string> DeploySmartContract(string, string, object[])
```

Fig: Deploying an Ethereum Smart Contract to a local node

Adapter Configurations

Knowing the address of the published smart contract(s), the access information to the connected systems, the contract interfaces on each part of the system landscape and the mapping information to transform different contract interfaces into each other (as part of the Template), the smart adapter configuration can be published to the Unibright Connector.

Depending on the defined role in the Workflow, the smart adapter either

- Defines an endpoint within the Unibright Connector, ready to be “called” (e.g. a Webservice)
- Defines an endpoint within the Unibright Connector polling a source (e.g. an FTP Server)
- Defines an endpoint within the Unibright Connector calling an external source (e.g. sending an IDOC or calling a smart contract function)

The different channels for different connection techniques are implemented in the source base of the Unibright Connector. The smart adapter configurations tell the system how to connect using one of the available channels.

```
try
{
    #region Custom Binding and Channel Factory
    CustomBinding binding = new CustomBinding();
    binding.Elements.Add(new TextMessageEncodingBindingElement(MessageVersion.Soap11, Encoding.UTF8));

    HttpsTransportBindingElement transport = new HttpsTransportBindingElement();
    transport.AuthenticationScheme = AuthenticationSchemes.Basic;
    transport.ProxyAuthenticationScheme = AuthenticationSchemes.Basic;
    transport.Realm = "XISOAPApps";
    binding.Elements.Add(transport);

    binding.SendTimeout = TimeSpan.FromSeconds(60);


    EndpointAddress address = new EndpointAddress(GetSetting(SettingKeys.DGVPI_ENDPOINTADDRESS));

    ChannelFactory<SI_OB_ASYNC_DELVRY> factory = new ChannelFactory<SI_OB_ASYNC_DELVRY>(binding, address);
    factory.Credentials.UserName.UserName = GetSetting(SettingKeys.DGVPI_USER);
    factory.Credentials.UserName.Password = GetSetting(SettingKeys.DGVPI_PASS);
    #endregion ↑ Custom Binding and Channel Factory

    SI_OB_ASYNC_DELVRY client = factory.CreateChannel();
    var d = new SI_OB_ASYNC_DELVRY1();

    try
    {
        DELVRY03 delivery = new DELVRY03();
        d.DELVRY03 = createDelvry03FromOpenStorageFile(action.PayloadIN, action.ActionID.ToString());


        await client.SI_OB_ASYNC_DELVRYAsync(d);

        ret.Success = true;
```

Fig: Sending an IDOC from the Unibright Connector to a SAP PI

Smart Query Sets

Knowing all adapter configurations, the Unibright contract interface and the template related specific interfaces, smart query sets are also generated by the Contract Lifecycle Manager.

It targets template specific questions, independently from the data source they belong to. Staying with the example of “Request for Quotation” such a question could be: “Show me all quotation results for product 123 that were fulfilled by supplier ABC”.

The Unibright Explorer handles this question (and the resulting graphical representation) via one place of accessing all relevant data: The Unibright Connector, querying transactions in a specific blockchain and performing queries on the connected systems, from the “old world” (e.g. FTP) to the new world of smart contracts.

```
public async Task<List<FilterLog>> queryTransactionsOfContract(string contractaddress, string abi)
{
    var contract = web3.Eth.GetContract(abi, contractaddress);
    var blocknumber = await web3.Eth.Blocks.GetBlockNumber.SendRequestAsync().ConfigureAwait(false);
    var ethFilterInput = new NewFilterInput();
    ethFilterInput.FromBlock.SetValue(new HexBigInteger(0));
    ethFilterInput.ToBlock.SetValue(new HexBigInteger(blocknumber));
    ethFilterInput.Address = new[] { contractaddress };

    var newEthFilter = await web3.Eth.Filters.NewFilter.SendRequestAsync(ethFilterInput).ConfigureAwait(false);
    var filterreturn = await web3.Eth.Filters.GetFilterChangesForEthNewFilter.SendRequestAsync(newEthFilter).ConfigureAwait(false);


    return filterreturn.ToList();
} ↑ Task<List<FilterLog>> queryTransactionsOfContract(string, string)
```

Fig: Querying a list of transactions of a Smart Contract

```
119         try
120         {
121             string FTPSERVER = SystemRepository.GetBusinessCaseSpecificSetting(SettingKeys.FTPSERVER.ToString(), BUSINESSCASE);
122             string FTPRENAME = SystemRepository.GetBusinessCaseSpecificSetting(SettingKeys.FTPRENAME.ToString(), BUSINESSCASE);
123             string FTPUSER = SystemRepository.GetBusinessCaseSpecificSetting(SettingKeys.FTPUSER.ToString(), BUSINESSCASE);
124             string FTPPASSWORD = SystemRepository.GetBusinessCaseSpecificSetting(SettingKeys.FTPPASSWORD.ToString(), BUSINESSCASE);
125
126
127             List<string> allFiles = PIPHelper.GetFileListFromFTP(FTPSERVER, FTPUSER, FTPPASSWORD);
```

PIPHelper.cs

```
159
160         public static List<string> GetFileListFromFTP(string FTPAddress, string username, string password)
161         {
162             List<string> files = new List<string>();
163
164             try
165             {
166
167                 //Create FTP request
168                 FtpWebRequest request = FtpWebRequest.Create(FTPAddress) as FtpWebRequest;
169                 request.Method = WebRequestMethods.Ftp.ListDirectory;
170                 request.Credentials = new NetworkCredential(username, password);
171                 request.UsePassive = true;
172                 request.UseBinary = true;
173                 request.KeepAlive = false;
174                 FtpWebResponse response = request.GetResponse() as FtpWebResponse;
175                 using (Stream responseStream = response.GetResponseStream())
176                 {
177                     using (StreamReader reader = new StreamReader(responseStream))
178                     {
179                         while (!reader.EndOfStream)
180                         {
181                             files.Add(reader.ReadLine());
```

Fig: FTP reading within the Unibright Connector

Summary and Outlook

This document gave an insight of the different components of the Unibright Framework and a walkthrough for the example-Usecase “Request for Quotation”: From the use-case related template, a basic integration workflow can be customized. Based on software architectural principles like Interfaces (“The Unibright Contract Interface”) all needed objects to run and monitor the ongoing business process are generated automatically.

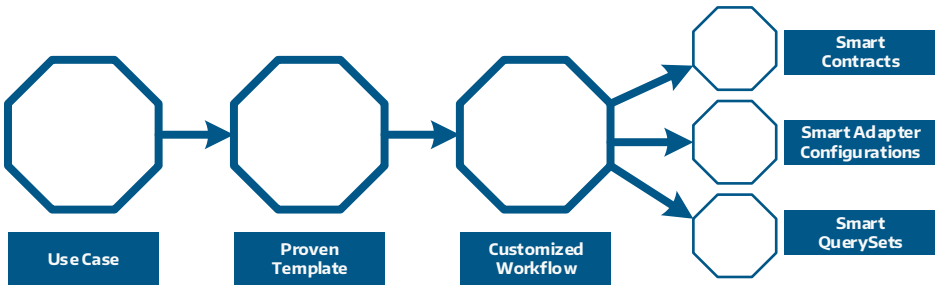


Fig: From use case to generated objects

For further information (not only about technical details) on the Unibright Framework:

- Visit unibright.io
- Get more insights on use cases in our blog (https://medium.com/@UnibrightIO)
- Download the whitepaper (https://unibright.io/files/Unibright_Whitepaper.pdf)
- Join the telegram group: https://t.me/unibright_io
- Get in contact with the team: team@unibright.io
- Meet us in person on conferences and meetups (see our website for details)