# Smart contracts using Blockly

Representing a purchase agreement using a graphical programming language

Tim Weingärtner
School of Information Technology
Lucerne University of Applied Sciences & Arts
Suurstoffi 41b, CH 6343 Rotkreuz
tim.weingaertner@hslu.ch

Rahul Rao
CSS Versicherungen AG
Tribschenstrasse 21, CH 6005 Lucerne
rahul.rao@stud.hslu.ch

Jasmin Ettlin
Suva Head Office
Fluhmattstrasse 1, CH 6002 Lucerne
jasmin.ettlin@stud.hslu.ch

Patrick Suter
Satisloh AG
Neuhofstrasse 12, CH 6340 Baar
patrick.suter@stud.hslu.ch

Philipp Dublanc
Lucerne Cantonal Hospital
Spitalstrasse, CH 6000 Lucerne
philipp.dublanc@stud.hslu.ch

*Abstract*— **This research addresses the issue that in-depth programming knowhow is needed to read and write smart contracts. The goal was making the creation of smart contracts accessible to non-computer experts by the use of a graphical programming language (Blockly). We used modularization to capture the complexity of legal contracts and developed a mapping process to transform the graphical representation to the smart contract programming language Solidity. We applied our approach to legal purchase agreements and proved the practicality of our solution and explored its limitations. A prototype was built to show the feasibility of our approach. Our industry partner challenged the prototype by applying it to the contract creation process. We consider our work as the first step towards an application of smart contracts in the non-IT world and outside the today's expert shaped ecosystem of blockchain specialists. Several continuative research questions have been derived from our finding and are listed at the end of this paper.**

***Keywords: blockchain; smart contracts; solidity; blockly; modularization***

## I. INTRODUCTION

Blockchain is considered a new, promising and versatile technology ([1]). In 2017 it gained broad attention due to the Bitcoin price rise and countless news articles describing its potential. Beside crypto currencies, smart contracts are a promising and powerful way to leverage blockchain technology. They allow two or more parties to agree on a contract with guaranteed execution upon the occurrence of predetermined events and direct access to a crypto currency. Szabo defined smart contracts as computer programs that replicate predefined contracts ([2]). Ethereum implements smart contracts with its own programming language Solidity. The program code is stored and executed on the Ethereum Blockchain using the Ethereum Virtual Machine (EVM).

Recent years have shown that using smart contracts by a wider user group hold different kinds of challenges:

- To implement a smart contract, profound programming knowledge and skills in the programming language is required. In the case of Ethereum this is Solidity. Non-computer specialists or untrained users with little programming knowledge have a hard time comprehending the contracts, interpreting their meaning and predicting the automated actions.
- Code errors or uncaught cases can lead to unforeseen executions and sometimes to the loss of large amounts of money. Once agreed upon a smart contract nobody can alter its execution. The DAO case has shown this in a dramatic way ([3]).
- Since the contracts are executed automatically and without enforcement through a third party, interactions with persons have to be incentivized and retained in the protocol used. Otherwise there might be deadlocks or endless waiting cycles. For example: Why should a buyer confirm the reception and thus execute the

payment if he already has received the goods? Therefore conventional legal contracts and their “execution protocols” cannot be transferred directly into smart contracts.

- The reuse of contractual code strongly depends on the structure of the smart contract and its level of generalization.

The aforementioned challenges have to be overcome to reach the goal of using the full potential of smart contracts for a broad application and outside the domain of information technology. With our research we investigated a possible solution for these problems using modularization and simplification: applying a graphical programming language to provide the access for non-computer specialists, modularization of proven code for reuse and addressing error susceptibility and predefined procedures to supply best practices in state protocols. As a development environment for the modularized code, the Blockly Framework from Google and its graphical programming language has been chosen ([4]).

Besides these technological challenges there are several legal as well as sociolegal questions, which we will also briefly touch in this paper.

This paper is structured as follows: First we describe our research process by explaining the underlying research question and sub-questions as well as the research design. We give a summary of existing approaches to modularize purchase agreements and provide a syntactical analysis of existing legal contracts. Than we describe the implementation of our proof of concept. This section is divided into a closer look in the modularization of the Solidity code and an explanation of the Blockly part. The conducted evaluation and their results round off the practical part. The conclusion summarizes the results and the future work gives an outlook to the next steps to be examined.

## II. Research Method

### A. Research Question

Based on the above identified problems, our research aims to answer the following question:

***RQ: Can a legal contract be modularized in a syntactically and semantically correct way so that a Solidity smart contract can be created automatically using building blocks and is the usage of such a procedure appropriate for non-computer specialists?***

This leads us to the following sub-questions which have been used to answer the research question RQ:

- ***SQ1: How does a legal contract have to be structured in order to be transformed into Solidity code?***
- ***SQ2: How can the individual components of a contract be mapped into Blockly templates as building blocks for smart contracts?***
- ***SQ3: Is it possible for a non-computer specialist to handle a smart contract using such a framework?***

Since there are a broad variety of contracts, this research focused on purchase agreements. They are well known and one of the major types of contracts. Our approach can be easily applied to other kinds of contracts like lending agreements or service agreements by adapting the code templates and their mapping to the graphical building blocks.

### B. Research Design

This research is based on the design science paradigm presented by Hevner et al. in [5]. Design science stands for constructing and evaluating artifacts to solve research questions. In information technology (IT) design science is a commonly used research methodology since explicitly applicable results are produced (programs). Therefore this method fits very well in our context. According to [6] design science includes the following six steps: problem identification, definition of the objectives for the solution, development, demonstration, evaluation, and communication.

During our research project two artifacts were developed: the design of Blockly templates to represent a purchase agreement and the concept for modularization of Solidity source code.

The evaluation of the artifacts is used to determine whether the research hypotheses derived from the research question can be confirmed ([7]). The metric we are using in this first attempt is a qualitative metric since we are evaluating our results with test persons picked from the target group of non-computer specialists but having legal background knowledge.

## III. Contract Analysis

### A. Related Work

Smith has described modularization of legal contracts in [8]. The concepts in his research have been derived from information technology and object oriented programming. Smith justifies the need for modularization in legal contracts with the reduction of complexity. High complexity in legal agreements today is a major drawback. Those contracts are modularized by using contractual boilerplate, which means the reuse of textual components. In [8] a variety of modular operators like: splitting, substitution, augmentation, exclusion, inversion and porting are described. Some of these operators are used in our research for modularization of the Solidity code as we will show later on.

Smart contracts are much more complex than legal agreements. While executing a purchase agreement the transaction of a physical object or commodity almost always has to be notified by humans. In the context of smart contract, this is a weak spot since each interaction between the blockchain and another system or person is a possible threat which cannot be covered by blockchain security.

There exist several concepts to implement a fair exchange protocol for the exchange or purchase of electronic data. Research is focusing on the question how a fair exchange can be realized without requiring a trusted third party (TTP). A general summary of those attempts can be found in [9] and [10]. Delgado-Segura et al. describe in [10] a fair exchange protocol based on the Bitcoin blockchain without the need of a TTP using smart contracts. All these approaches focus on electronic commerce (e-commerce) where the goods are data that can be delivered gradually or can be encrypted.

Approaches where physical goods are traded using smart contracts are rare. OpenBazaar[1] is such a platform. Outside the blockchain world ebay comes close to this kind of trading. Both platforms offer a TTP in case of a dispute.

Research in the legal domain concerning smart contracts have been conducted e.g. in [11] or [12]. Both articles focus on the problems using smart contracts in an existing legal area and the conflicts between legal practice and technological possibilities. In our research we encountered some of the conflicts described in the articles above like missing international acceptance or possible conflicts with applicable law. These are fundamental legal problems, which cannot be solved by science. To solve them a political and legal discussion is needed.

A specific examination in replacing paper contracts by smart contracts can be found in [13]. Egbertsen et al. come to the conclusion that it is possible to transform paper contracts into smart contracts. At the same time they highlight some weak spots which are mainly addressing privacy issues and the existing complexity in contractual clauses.

## B. *Purchase Agreement*

As a starting point for our research we analyzed several purchasing contracts ([14], [15], [16], [17], [18]) to distinguish the relevant elements for modularization of a legal and written purchase agreements (see Table I)

Since for a smart contract there is no TTP like a court, jurisdiction in the common sense does not apply to it. It is rather inherent in the system. Including a TTP would only apply if a special condition would be implemented like transforming special rights in the case of a dispute. In a first version (V1) we modeled a simple contract without payment details, warranty, jurisdiction and special agreements. We introduced financial incentives in order to guarantee satisfaction for the two involved parties. These incentives are represented by the fact that both parties pay twice the purchase amount as a deposit in the form of Ether in the smart contract. As a result, both parties have the same incentive to abide by the contract. This money deposit implements an approach, which Szabo proposes in 1994 in his Digital Cash Concept [2]. He describes the possibility of a token, which can serve as a digital currency, comparable to today's crypto-currencies. Szabo explains that this token alone is not enough to secure a transaction. There is a need for further mechanisms, for which the above-mentioned protocol, with the monetary deposit as an incentive, can serve as an example. The disadvantage when both parties have to pay double the purchase price as a deposit is that depending on the purchase price very large sums must be available and are bound during the transaction.

TABLE I: CONTRACTUAL ELEMENTS

| Contractual Element | Description |
|---|---|
| Contracting parties | Mostly there are two contractors, buyer and seller. |
| Purchase item | The item offered for sale. |
| Purchase price | The amount to pay for the item of purchase. |
| Payment | Payment details. |
| Delivery | Information about the delivery of the object of purchase. |
| Warranty | Warranty information in the event of damage or wrong orders. |
| Special agreements | Free text for the capture of any additions. |
| Jurisdiction | Information about the court in case of disputes. |
| Signatures | Signatures of the contracting parties, respectively of the guardian. |

In a second version (V2) we modeled a more complex contract with payment details and predefined special agreements. V2 is based on a sales contract of Satisloh AG. Satisloh AG is one of the leading machine manufacturers for ophthalmic and precision optics manufacturing. This contract has special payment conditions, delivery and installation options.

Depending on the customer, Satisloh can decide which payment option to choose. The customer can either pay in three payment rates, upfront or after the delivery has been carried out. It is also possible that the customer decides to pick up the machine himself or have his own carrier of choice pick it up. Besides that it is possible for Satisloh to abort the contract if required. Therefore the corresponding contract option has to be added to the smart contract. Another option that can be added to the contract is the possibility that a Satisloh expert installs the machine for the customer.

Two parties apply as contractors for both versions (V1 and V2). Swiss law does not require a written signature for a so-called “Fahrniskauf”, a purchase other than property purchase. Legal details can be found in [19]. The contract is concluded by the acceptance of the buyer and becomes valid in the blockchain. A special process is needed for delivery. There is a significant difference between a pick up by the buyer and a delivery by a carrier. Figure 1 shows the state chart for V1 with a carrier. Figure 2 shows the extended state chart for V2, the Satisloh machine contract with a carrier.

[1] http://www.openbazaar.org

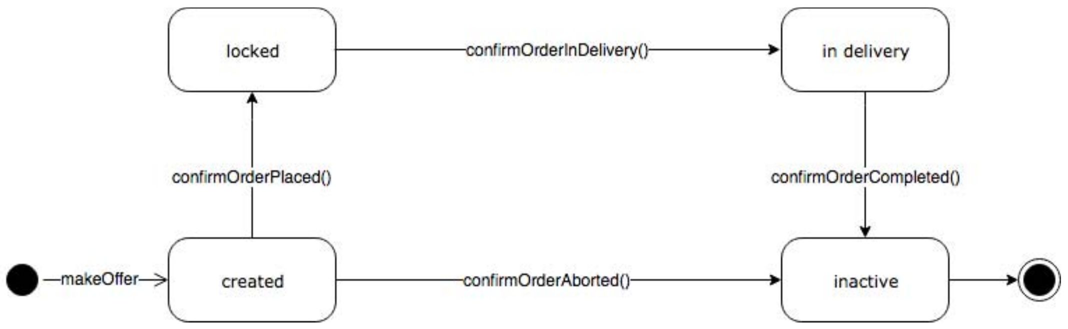


Figure 1: State chart of V1 with carrier

The carrier introduces another player into the process. This new player is a potential risk that has to be considered.

## IV. Implementation

As described in the research design the research question (RQ) is addressed with a software artifact. Therefore to prove the research hypotheses derived from RQ and to verify the usability of the concept, a contract development environment (CDE) has been developed. The realized CDE is running in a browser environment using the Blockly Framework embedded into HTML and Javascript code. For our implementation, the following concepts and decisions apply:

- The foundation for our contracts is the Safe Remote Purchase from the Solidity website [20]. Of central interest was the concept of using incentives to get both parties to stick to the contract.
- The different components of the Solidity contract have been transformed directly into Blockly. This keeps the programming effort for the Blockly generator as low as possible.
- In Blockly and Solidity, not all the attributes from Table I are used, since it is technically not necessary. The more data there is in the smart contract, the more transaction costs (gas) has to be paid for each function call. This reduction can potentially be a risk, since anything not included in the smart contract and therefore not inserted into the blockchain cannot be protected against malicious changes.
- For simplicity, we chose the crypto-currency Ether for payments. It would be possible to use state-issued currencies (Fiat money) such as Swiss francs too. This can be integrated into the entire system with the help of Oracles, which are services submitting information about real-world events to a blockchain to be used by smart contracts. Details can be found in [21].
- For the first implementation the value of an object is stored as an integer in the smart contract since doubles are not fully supported by

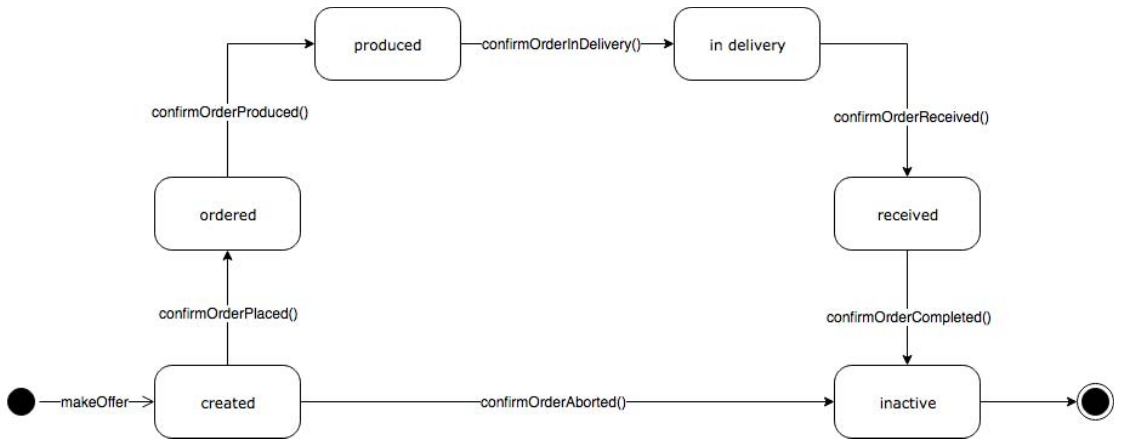


Figure 2: State chart of V2 with carrier

Solidity. In consequence, in Blockly you can only use integers for the price. Currently we are using Ether as currency but the extension of the prototype with sub-currencies of Ether like Wei, Kwei or Szabo is trivial.

- The ID of the purchased item or good is stored as a comment in the smart contract, since this ID does not have a contribution to the process.
- We use events to alert the carrier to the order. How these are intercepted in real world cases and further processed by the carrier is not part of this work.

## *A. Modularization in Solidity*

This section shows how the smart contract in Solidity has been modularized. As mentioned the Safe Remote Purchase contract was derived from the Solidity website [20]. The contract is called "Safe Remote Purchase" as it not only covers the steps of entering a purchase agreement, but it also ensures by means of financial incentives that the buyer pays the goods and the seller really hands over the goods. The deposit is refunded at the end of the contractual process.

The class diagram in Figure 3 shows the modularization of the Solidity code of V1. On the one hand we work with abstraction, on the other hand with typing. Abstraction is used for the delivery variants: with and without a carrier. Both contractual types have certain functions in common. The two child classes "DeliveryContract" and "PickupContract" inherit from the superclass "BaseContract". In the legal context this is called substitution by Smith ([8]). The two classes also use enumerations to represent their status. In Solidity, enumerations defined in a superclass cannot be overridden or augmented by a derived class. For this reason, the status is defined in the child classes. As an additional case, a class "Options" was introduced. As an example the functionality of the cancellation of the sales was modeled as optional. The abortion of the agreement results in the cancelation of the whole smart contract.

Version V2 implementing the Satisloh purchase agreement is designed accordingly. The super class "SatislohBaseContract" is extended by three child classes "ThreeInstallmentPaymentContract", "Upfront PaymentContract" and "AtEndPaymentContract" to represent the possible payment conditions. In addition, there is again the class "Options". This class defines additional

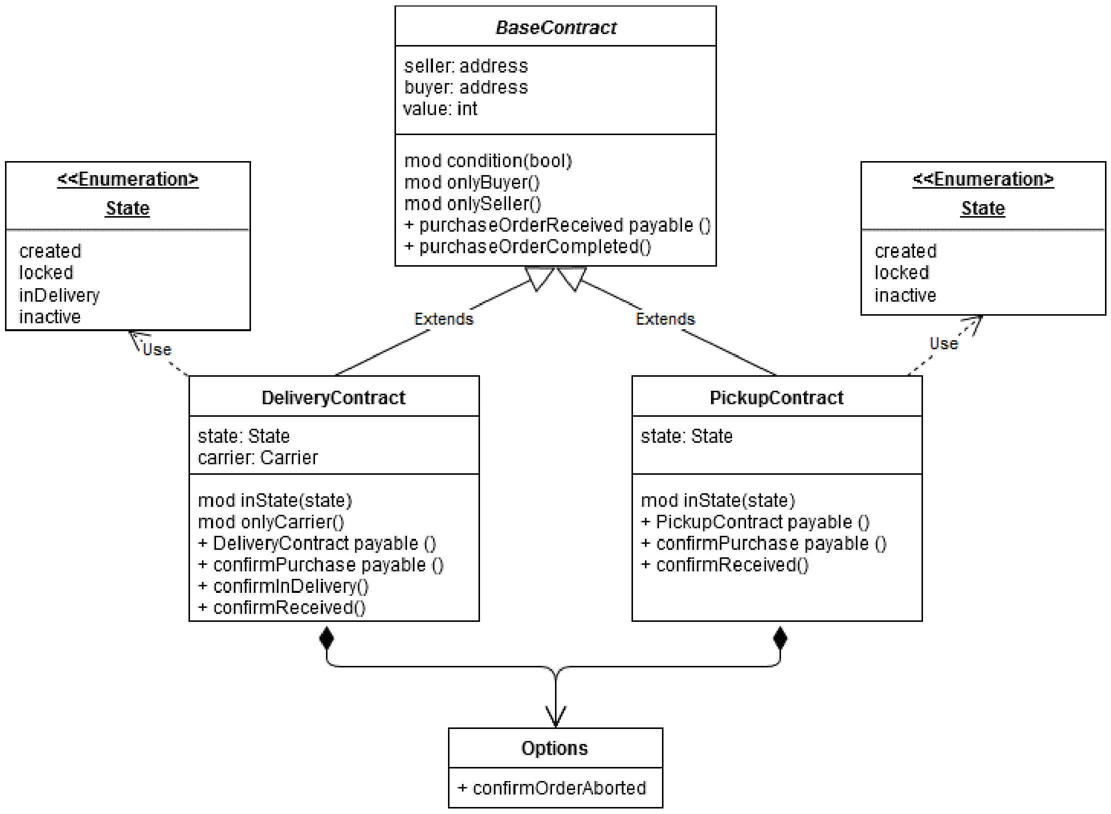


Figure 3: Class diagram for modularization of contract V1

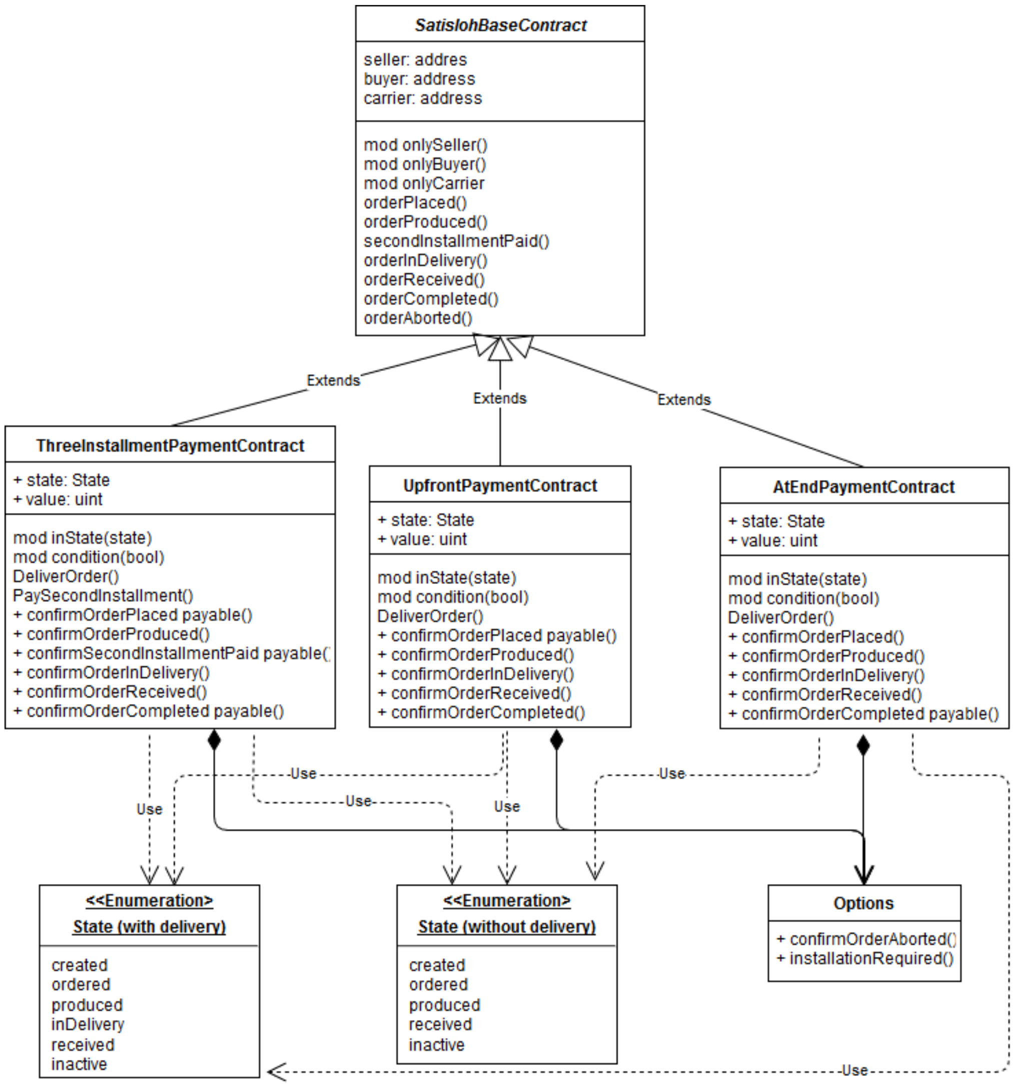


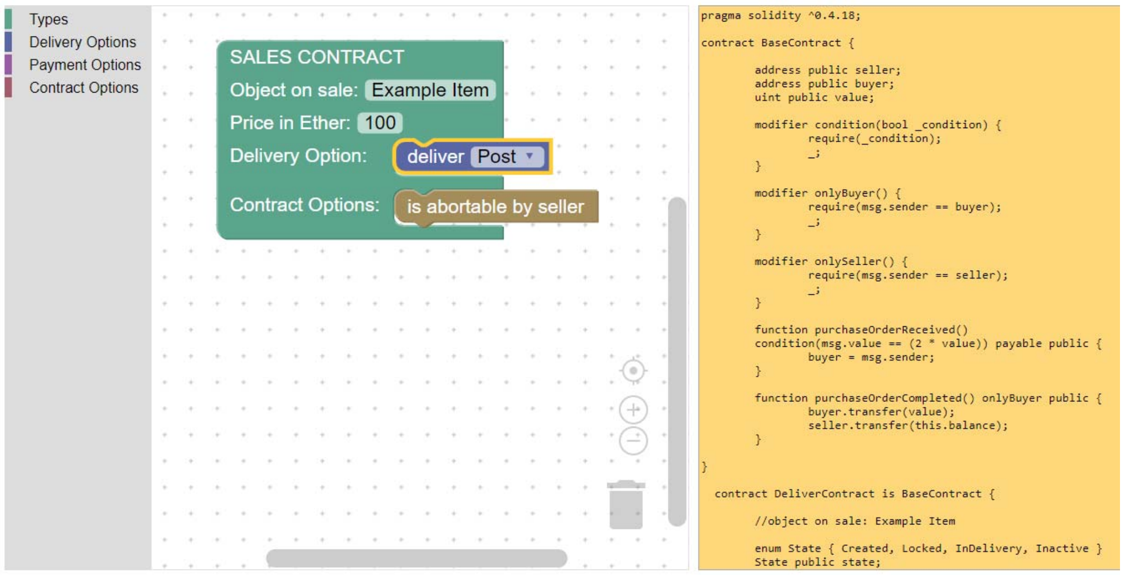
Figure 4: Class diagram for modularization of contract V1

functions that can be used. In case of V2 on the one hand, there is the function of the contract abortion and on the other hand, the decision whether Satisloh is the installer or not. At last there are the two enumerations, which represent the available states. The class diagram is shown in Figure 4.

## B. *Modularization using Blockly*

The Blockly framework was chosen as development environment since it is used in several cases to introduce non-computer specialists to a programming language. MITs Scratch language is one example of this. Here children learn to program without the need of first learning the syntax of a programming language. In [22] the advantages of using this kind of programming language are explained. Our research hypothesis derived from SQ3 is, that using a Blockly based framework will make it possible for people without an IT background to understand smart contracts and define own agreements.

The class diagram in Figure 3 forms the foundation of the modularity in Solidity and therefore the blocks in Blockly. It was explicitly ensured that no Solidity knowledge must be present in order to create a smart contract with Blockly. Furthermore, some contractual attributes, such as name, first name, address or detailed description of the purchased item were intentionally omitted since this information was not necessary for the smart contract. This information is profile information that belongs on the website of the sales platform and not necessarily in the smart contract. It was decided that


Types
Delivery Options
Payment Options
Contract Options
SALES CONTRACT
Object on sale: Example Item
Price in Ether: 100
Delivery Option: deliver Post
Contract Options: is abortable by seller

```
pragma solidity ^0.4.18;

contract BaseContract {

        address public seller;
        address public buyer;
        uint public value;

        modifier condition(bool _condition) {
                require(_condition);
                _;
        }

        modifier onlyBuyer() {
                require(msg.sender == buyer);
                _;
        }

        modifier onlySeller() {
                require(msg.sender == seller);
                _;
        }

        function purchaseOrderReceived()
        condition(msg.value == (2 * value)) payable public {
                buyer = msg.sender;
        }

        function purchaseOrderCompleted() onlyBuyer public {
                buyer.transfer(value);
                seller.transfer(this.balance);
        }

}

  contract DeliverContract is BaseContract {

        //object on sale: Example Item

        enum State { Created, Locked, InDelivery, Inactive }
        State public state;
```

Figure 5: Example contract V1 in the Blockly environment

only the ID of the buyer and seller, the price, as well as the ID of the purchased item should be stored in the smart contract and thus stored in the blockchain. All necessary information can be derived from these IDs.

The smart contracts are assembled in Blockly with the help of three component groups: the *types*, the *delivery options* and the *contractual options*. A *type* consists of exactly one *delivery option* and several optional *contractual options*.

*1) Types* represent the shell of the smart contract and provide the source code for the BaseContract. For V1 the type of a "Sales contract" has been defined as block with two attributes: an editable field for the name or the ID of the purchased item as well as an editable field for the selling price in Ether (see Figure 5).

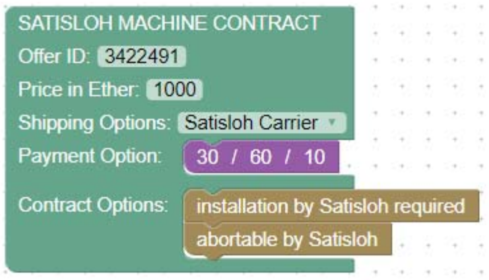

SATISLOH MACHINE CONTRACT
Offer ID: 3422491
Price in Ether: 1000
Shipping Options: Satisloh Carrier
Payment Option: 30 / 60 / 10
Contract Options: installation by Satisloh required
abortable by Satisloh

Figure 6: Example contract V2

*2) Delivery options* depict the ways the item reaches the buyer. On the one hand the object can be picked up (block "pick up") and on the other hand it can be delivered (block "deliver"). It is up to the seller to define which option is chosen since only one per contract is possible. At the same time, the choice of the delivery option determines which source code is generated for a "DeliveryContract" or a "PickupContract" (see Figure 5). Using the block "deliver", the seller can define which carrier should be chosen for delivery. Two examples have been implemented: "Post" or "DHL".

*3) Contractual options* offer the creator of the smart contract the opportunity to expand the functionality. Several options can be defined and combined per smart contract. For a first proof of concept the option "is abortable by seller" was created (see Figure 5).

Figure 6 shows an example of V2, the Satisloh machine contract. The offer ID is for documentation purposes only. The three payment options have been realized as blocks symbolizing the payment milestones: 30/60/10 for the "ThreeInstallment PaymentContract", 100/0/0 for the "UpfrontPayment Contract" and 0/0/100 for "AtEndPaymentContract". V2 also shows the combination of two *contractual options*. The order is not relevant for the correctness of the resulting Solidity code.

## V. Evaluation

### A. Evaluation setup

As a test environment for the Solidity code the integrated development environment (IDE) Remix [2] was used. The Remix IDE provides a blockchain in a sandbox to perform simple functional tests.

The research team tested version V1 as a proof of concept. The aim was to make corrections of technical errors and reasoning errors that were not recognized before the concrete implementation of the Solidity code of a class.

The evaluation of V2 first was conducted with a graduate of the master's degree in law. The evaluation was conducted as follows: firstly she was introduced to the use case, as well as the principles of smart contracts and Blockly. Subsequently, she was encouraged to use the application as described. Afterwards she was interviewed using a questionnaire.

As a second test person for V2 the internal lawyer of Satisloh was selected. The system had to meet higher requirements since, as a lawyer he knew all elements of the Satisloh contract in detail. He knew which functionalities had to be covered in order for the use case to correspond to the real world conditions. Therefore, more critical feedback was expected in this case. The evaluation procedure was identical to the previous one, whereby the focus was clearly placed on the criterion usability and correctness from a legal perspective.

### B. Evaluation results in detail

Table II at the end of this paper shows the results of the empirical questions in a summarized version for both test persons. Several aspects have been covered by the evaluation procedure:

- Usability aspects from the view of non-computer specialists.
- Completion of given tasks.
- Assessment from a legal perspective which has been especially valuable since the test persons both have a legal education background.
- Overall evaluation of the CDE.
- Assessment of the practicability for Satisloh which could only be answered by the internal lawyer.

### C. Summary of the evaluation

In summary, the following findings can be drawn from the evaluation:

All test persons found their way around the solution very quickly. The design and intuitive handling in Blockly is easy to grasp and a welcome support for non-computer specialists. In addition, the solution can cover the minimum requirements, which are needed for the implementation of a legally valid purchase agreement.

On the other hand the evaluation has also shown some drawbacks. The practical suitability is not quite given by the internationally different legal situation and the current lack of recognition of digital documents. Furthermore, the handling of crypto-currencies as a means of payment is not a viable option for many companies, so that conventional currencies are still the standard. The lack of integration of individual adjustments in the current solution has been perceived as an obstacle. Attributes such as liability, warranty and warranty processing are not yet offered by the current solution. We are confident that this drawback can be addressed by our further research.

## VI. Conclusion

Our research has shown the high potential of a graphical programming language like Blockly for the implementation of smart contracts. From a legal perspective there remain limitations and restrictions.

However the research sub-questions formulated in chapter II could all be answered:

***SQ1: The class design in Figure 3 and 4 show a possible structure of the Solidity code to represent a legal contract.***

***SQ2: The mapping between the Solidity code and the Blockly objects is part of our CDE. A complete description of the mapping had to be omitted due to the limitations of space.***

***SQ3: The conducted evaluation and the following interviews show the practicability of the CDE and the complete solution.***

In summary the research question RQ can be answered as follows:

***Yes, a legal contract can be modularized in a syntactically and semantically correct way so that the building blocks can be transformed into a Solidity smart contract. The only limitations are complex, non-standardized contract conditions.***

***Yes, non-computer specialists can handle such a system. Blockly has proven its worth and is a great way to bring smart contracts closer to non-computer specialists.***

The blockchain technology and crypto-currencies are still far from being trusted and accepted in a business environment. First approaches like this research exist and once the legal hurdles are taken, solutions will come into production very fast. All eyes are on first movers who can prove that this could be a viable option for the future of law.

[2] http://remix.ethereum.org

## VII. Future Work

Our first implementation and tests have shown the high potential for further research in this area. The following questions have been identified:

*1) Exchange and return:* The blockchain stores transaction data irreversibly in a database. In the Swiss Law (OR, Article 40a ff.) the right of withdrawal is clearly regulated. It is also a service of many companies to increase customer satisfaction. Smart contracts have to address this right.

*2) Dunning process:* In case of late delivery a dunning process has to be installed. A dunning for the payment is not needed since the money is reserved from both partners by entering the contract. Moreover a dunning process for conferment and delivery has to be installed.

*3) Oracles:* Oracles offer a link to functionality outside the blockchain. They can be used for the integration of banking service providers or credit card companies. However, through this integration an advantage of the blockchain is lost as there is a third party, which interacts between the end user and the blockchain.

*4) Contract elements:* It has been decided for the presented implementation that due to the increasing transaction costs, not all attributes of the contract are mapped in the smart contract. However, as this extension would be closer to practice, this could be further elaborated in another work.

*5) Crypto-currencies:* As long as crypto-currencies are not officially recognized on the market, an intermediate solution with Fiat money should be made possible in order to promote the use of smart contracts. The actual volatility of crypto-currencies is another obstacle for legal contracts.

*6) International acceptance:* Digitalization progress varies from country to country. It will take some time before a smart contract is internationally accepted. This is a research task for the legal domain.

## Acknowledgment

The authors would like to thank Satisloh AG for the support during creation and evaluation of contract V2 as well as for the valuable feedback. Especially the two test persons have given extremely valuable inputs and impulses for our research.

## References

[1] A. Rosic. "5 Blockchain Applications That Are Shaping Your Future". Retrieved from <https://www.huffingtonpost.com/ameer-rosic-/5-blockchain-applications_b_13279010.html> (2016, November 28) Date accessed: 02/10/2018

[2] N. Szabo. "Smart Contracts", Retrieved from <http://www.fon.hum.uva.nl/rob/Courses/InformationInSpeech/CDROM/Literature/LOTwinterschool2006/szabo.best.vwh.net/smart.contracts.html> (1994) Date accessed: 10/22/2017.

[3] V. Buterin: "Critical update re: DAO vulnerability". Retrieved from <https://blog.ethereum.org/2016/06/17/critical-update-re-dao-vulnerability/> (2016, November 17) Date accessed: 02/10/2018.

[4] Google Developers. "Introduction to Blockly". <https://developers.google.com/blockly/guides/overview> (2016, June 23). Date accessed: 12/27/2017.

[5] A. R. Hevner, S. T. March, J. Jinsoo, S. Ram. (2004). "Desicn Science in Information Systems Research".MIS Quaterly, 28(1), pp. 75–105, March 2004.

[6] K. Peffers , T. Tuunanen , M. A. Rothenberger, S. Chatterjee. "A Design Science Research Methodology for Information Systems Research", Journal of Management Information Systems, 24:3 (2007), pp. 45-77

[7] S. T. March, G. F. Smith. "Design and natural science research on information technology." Decision support systems 15.4 (1995), pp. 251-266.

[8] H. E. Smith. "Modularity in Contracts: Boilerplate and Information Flow". 104 University of Michigan Law Review 1175, 2006.

[9] H. Pagnia, H. Vogt, F. C. Gärtner. "Fair Exchange". The Computer Journal, Volume 46, Issue 1, pp. 55–75, January 2003.

[10] S. Delgado-Segura, C. Pérez-Solà, G. Navarro-Arribas, J. Herrera-Joancomartí. «A fair protocol for data trading based on Bitcoin transactions". Future Generation Computer Systems, 2017, in press.

[11] M. Giancaspro. "Is a 'smart contract' really a smart idea? Insights from a legal perspective". Computer Law & Security Review, vol. 33, no. 6, December 2017, pp. 825-835, Elsevier.

[12] K. E. C: Levy. "Book-smart, not street-smart: blockchain-based smart contracts and the social workings of law." Engaging Science, Technology, and Society 3 (2017) pp. 1-15.

[13] W. Egbertsen, G. Hardeman, M. van den Hoven, G. van der Kold, A. van Rijsewijk. "Replacing Paper Contracts With Ethereum Smart Contracts." (2016).

[14] TCS. "Kaufvertrag". Retrieved from <https://www.tcs.ch/mam/DigitalMedia/PDF/Booklets/Kaufvertrag.pdf>. Date accessed: 10/24/2017.

[15] Comparis. "Auto-Kaufvertrag". Retrieved from: https://www.comparis.ch/carfinder/info/glossar/kaufvertrag>, Date accessed: 10/24/2017.

[16] büez web services GmbH. "Kaufvertrag Vorlage Gratis Muster". Retrieved from: <http://www.conviva-plus.ch/?page=906>, Date accessed: 10/24/2017.

[17] Muster-Vorlage.ch. "Kaufvertrag Vorlage Schweiz". Retrieved from: <https://mustervorlage.ch/kaufvertrag-vorlage/>, Date accessed: 10/24/2017.

[18] Anwaltsbüro Schmid Heinzen Humbert. "Tierkauf". Retrieved from: <http://www.tierrecht.ch/tierkauf.html>, Date accessed: 10/24/2017.

[19] Bundesversammlung der Schweizerischen Eidgenossenschaft. "OR Art. 11." (1911, March 30). Retrieved from <https://or.gesetzestext.ch/artikel.cfm?key=12&art=Die_Entstehung_der_Obligationen>, Date accessed: 10/15/2017.

[20] Solidity. "Safe Remote Purchase". Retrieved from <https://solidity.readthedocs.io/en/develop/solidityby-example.html#safe-remote-purchase>, Date accessed: 11/23/2017.

[21] P. Zhang, J. White, D. C. Schmidt. "Design of Blockchain-Based Apps Using Familiar Software Patterns to Address Interoperability Challenges in Healthcare." Vanderbilt University, Nashville, TN. Retrieved from <http://www.dre.vanderbilt.edu/~schmidt/PDF/PLoP-2017-blockchain.pdf>, Date accessed: 02/02/2018

[22] M. Resnick, J. Maloney, A. Monroy-Hernández, N. Rusk, E. Eastmond, K. Brennan, A. Millner, E. Rosenbaum, J. Silver, B. Silverman, Y. Kafai. "Scratch: Programming for All". Communications of the ACM. vol. 51, no. 11, November 2009.

TABLE II: EVALUATION RESULTS

| Test Aspect | Test person 1 | Test person 2 |
|---|---|---|
| Finding the required blocks and their meaning. | Since the blocks are highly aggregated there was no problem in finding the needed building blocks. | After a short introduction the legal counselor succeeded in finding all necessary blocks.<br>He criticized that the blocks are all arranged at the same level in Blockly. It would be more logical if the contract type would be located at a higher level and the contract components as a subgroup of the respective contract type<br>The individual blocks are understandable and fit to the standard contract according to the terms and conditions of Satisloh AG. |
| Handling of the blocks. | The handling was very intuitive. Wrong positioning is technically prohibited. | The handling was intuitive. |
| Assemble a given test contract. | A given testcase was completed without error on the first attempt. | A given testcase was completed without error on the first attempt. |
| Correctness from a legal perspective. | To be valid, the contract must make it clear that both parties give their mutual consent. The object of purchase must also be specified.<br>Willingness comes from the seller by the placement of the offer and by the buyer upon acceptance of the offer. Another criterion is that the price is determinable. In the solution, this is fulfilled by explicitly defining the price in ether. The contract would therefore be legally binding. | The smart contract ensures contract execution, especially the cash flow. However, the cash flow is only a small part of a contract.<br>Currently the smart contract is not seen as a substitute for a legal contract. A legal contract will also cover issues such as liability, warranty, guarantee processing, etc. which is not part of the smart contract. |
| General evaluation of the solution and the CDE. | The solution is easy to handle and is well structured from a design perspective. A potential user of this system does not need an intense training.<br>It is questionable if more complicated types of contracts including several specific agreements (e.g. guarantee, special conditions) can be handled in a similar easy way. | The smart contract might be suitable for highly standardized contracts and their settlement. This is probably more likely the case for large companies or machine-to-machine contracts. If smart contracts become practicable, a solution for easy creation of these contracts is needed, since lawyers do not usually have the necessary skills to program a contract. The presented solution using Blockly is a valid variant to generate the necessary code. |
| Evaluation of the added value for Satisloh. | | The practical benefit for Satisloh AG is rather low. The main reason for this is that the standard contract is the exception. Most often the standard contract is used and then heavily modified in order to satisfy the customer needs. In addition, neither Satisloh nor its customers use crypto currencies.<br>The international legal situation is considered problematic. The authorities of many states do not even accept electronic documents as legally binding. It will take some time until a smart contract is accepted internationally, which is essential for an international company like Satisloh AG. |