COOPERATIVE RECONFIGURATION OF USER INTERFACES FOR LEARNING CRYPTOGRAPHIC ALGORITHMS

BENJAMIN WEYERS, WOLFRAM LUTHER
Dept. of Computer and Cognitive Science, U. of Duisburg-Essen, Lotharstr. 63
47057 Duisburg, Germany,
{weyers, luther}@inf.uni-due.de

NELSON BALOIAN
Dept. of Computer Science, U. de Chile, Blanco Encalada 2120,
Santiago, Chile,
nbaloian@dcc.uchile.cl

Cooperative work in learning environments has been shown to be a successful extension to traditional learning systems due to the great impact of cooperation on students’ motivation and learning success. In this paper we describe a new approach to cooperative construction of cryptographic protocols. Using an appropriate formal language for modeling user interfaces, students describe a protocol step by step, modeling subsequent situations. The system automatically generates a colored Petri subnet, which is matched against an existing action logic specifying the protocol, thus allowing formal validation of the construction process.

Keywords: cooperative construction, cryptographic algorithm, learning environment, dialog and interaction logic modeling.

1. Introduction

Cryptographic protocols play a central role in computer applications and the World Wide Web. Trading and banking applications, for instance, require a secure communication infrastructure, which cannot exist without cryptographic protocols. Thus, the teaching of cryptographic protocols, algorithms and frameworks using modern hardware like tablets or PDAs is continuously gaining in relevance and has become an integral part of computer studies. In this context, cryptographic protocols mainly deal with the private exchange of information or sharing of distributed keys for encrypted communication over unsecure communication channels.

Often, cryptographic algorithms describe communication between various partners, as is the case in key exchange protocols. Besides the two communication partners (“Alice” and “Bob”), a third participant is often involved, also called a trusted third party (TTP), which shares a secret key with Alice and Bob and is thus trusted by them. Teaching such distributed and complex processes for the exchange of information has already been investigated using a distributed learning system called CoBo [21]. This learning application combines a distributed infrastructure involving up to four work stations, a complex algorithm visualization component and a software implementation, concept keyboards (CK). Evaluation of students’ success demonstrated the efficiency of this
distributed learning system [36], resulting in the collaborative extension described in [45]. This collaborative extension was evaluated in further work [48] and has shown the high impact on cooperation and collaboration of using CKs in the learning of cryptographic protocols. A CK is a user interface with a reduced set of keys that trigger more complex actions in the context of the task being accomplished by the system [3, 5]. Thus, a given concept represents the functionality paired with the effect of the operation on the state of the protocol (or algorithm, etc.) controlled by the CK. This kind of association of concepts is fostered by aspects of design psychology, such as the rule of closeness [31, 38].

However, by creating input interfaces, the CK approach addressed the development of a mental model [19] of the process flow only implicitly [45]. Our running example was a key exchange protocol developed by Needham and Schroeder (NSP) in [29] that involves three different participants: the two communication partners and a TTP. A problem of the first published version was an eventual attack by a man in the middle. This weakness was fixed by extending the basic protocol. The idea of the former approach that participants had to assign a set of buttons to specific roles in the NSP and then to simulate the protocol using the interface they had created. In that approach, any given button represents one specific operation in the protocol that can only be executed by a specific participant in the protocol in a specific situation or step. Combining the operation with the participant and its execution was the cognitive effort the students had to perform. In performing this task, students who worked cooperatively achieved a better understanding of the protocol than students who worked individually to create one part of the protocol [48]. The students learned to execute the operations through distributed simulation of the protocol using the created user interfaces. Each operation was executed in the situation at the moment in which it was provided in the protocol. Still, the main learning objective was how to assign the operations to the participants and not to understand the protocol step by step or to understand the function of the steps themselves, like creating a certain message, encrypting it, and sending it to a communication partner.

The paper at hand reports on an extended approach to the learning of cryptographic protocols using a new technique for reconfiguring user interfaces. It extends the basic approach of using CKs to include an individual perspective as well as the cooperative approach described in [45]. In the latter, an interface was cooperatively created using an interactive editor for modeling user interfaces based on formal languages. The work at hand uses reconfiguration techniques for creating the protocol based on a formal approach to modeling user interfaces. Here, reconfiguration means that an interface offering basic elements for interaction is configured step by step. By applying reconfiguration operations, the interface’s logic describes the protocol, which will be explained below. If errors are detected in the protocol using verification and validation techniques based on the formal model, further reconfiguration operations can be applied to the user interface to correct the protocol.

A variety of verification and validation tools can be applied to a protocol created through user interface reconfiguration, thus checking the student’s solutions for
Cooperative Reconfiguration of User Interfaces for Learning Cryptographic Algorithms

correctness according to an expert model. This new concept of user interface reconfiguration will be embedded into a collaborative construction of personal interfaces to achieve the benefits of cooperation in learning cryptographic protocols based on the concept first described in [47]. A concluding simulation of the protocol by involved partners can demonstrate the overall functionality of the created interfaces and effect the communication between the participants, resulting in complete synchronization of the process and the mental models.

2. Related Work

Cooperation in learning has already shown an important impact on achieving learning success [30, 42]. The work in small groups offers opportunities for the students to share insights and observe each other’s strategies for solving a given problem and thereby to learn from each other [7, 2]. Former research and development in our group [4, 5] together with case studies and evaluation experiments [3] have shown the positive impact of cooperative learning in the context of learning complex algorithms and protocols. The cooperative creation of a user interface for distributed simulation of cryptographic algorithms has especially shown the positive impact of cooperation to learning success [48] based on the approach described in a former work [45].

In the learning of complex algorithms, systems and frameworks for the visualization and animation of algorithms play a central role. Since 1988 many algorithm visualization systems have been developed. In [39] Stasko et al. give an overview of software and algorithm visualization and their influence on developing and understanding complex algorithms. In [10] Diehl et al. also cover aspects of algorithm animation and its impact on software development processes in contrast to visual programming and diagramming of software design. Various repositories of software visualization and algorithm animation exist all over the Web; one example is the “Algorithm Animation Repository” [9]. In the context of this work, Cattaneo et al. [8] describe algorithm visualization for cryptographic protocols. Kerren, Müldner, and Shakshuki describe solutions for algorithm visualization and explanation systems [18] as well as for Web environments based on hypertext languages [37]. Basic research has also been conducted by Eisenberg [13] and others on algorithm visualization. Works like that of Archer [1] describe how human beings use visual artifacts and languages to make visible their cognitive thoughts and worlds and thus formulate their mental model of a complex environment. This is particularly important for learning environments that focus on the reflection of learned knowledge and therefore the materialization of abstract concepts and mental models as is the case of the learning process described in the work at hand [26].

The adaptation of user interfaces is a familiar topic in various research areas of studies in human-computer and human-machine interaction. An adaptive user interface is described by Langeley in [25] as an “interactive software system that improves its ability to interact with a user based on partial experience with that user.” The impact of adaptive interfaces on human cognition has been shown by Navarre et al. In [28] and [27]
the authors describe the use of adaptation techniques in safety critical applications, especially the design and reconfiguration of airplane interfaces.

The reconfiguration of user interfaces when learning cryptographic protocols have shown to be a promising approach in cooperative learning [48], but still needs defined formal and semi-formal approaches to the adaptation of user interfaces. De Rosis et al. propose in [35] a formal approach to synthesizing user interfaces from formally described interaction based on Petri nets (PNs). Using PN-based modeling languages, various analysis techniques, algorithms and approaches for net transformation can be utilized for reconfiguration. Ehrig et al. [11, 12] describe formal approaches for net transformation based on PNs and category theory. In [14] an overview can be found of how PNs can be used to describe interfaces and interaction. PNs were first introduced in the dissertation of Carl Adam Petri in 1962 [32]. Since then, many different types of extension have been developed on the basis of Petri’s formalism. Various works, like that of Baumgarten [6] or Priese [33], describe different types of PNs. In [17, 16] Jensen et al. described colored PNs that introduce complex types of tokens that can be compared to object-oriented data types paired with a complex inscription language for places and transitions in a PN. Another extension, which introduces synchronous channels and reference mechanisms to PNs, is described by Kummer [24].

Besides modeling user interfaces and their behavior on a formal level, validation techniques and tools are also highly relevant. In learning systems, verifying a student’s solution is a major element in giving the student advice in order to foster learning success. One specific example of how PNs can be used for validation in bioinformatics is offered by Koch et al. in [20]. The authors describe the use of Petri net formalism in the modeling and validation of breakdown pathways. In [34] Ráth et al. propose interactive validation techniques through simulation for validating PNs. Techniques based on iterative prototyping also deal with this problem, as described in [40]. Fronk et al. outline in [15] an approach to state space analysis for PNs on the basis of relation-algebraic methods that can be used for automatic verification of the created PNs.

3. Formal Reconfiguration of User Interfaces

In this section we will describe the cooperative construction of cryptographic protocols using a formal approach to reconfiguring user interfaces as a learning process. This formal approach is based on (a) a formal modeling language for the interaction-logic of a user interface and (b) on a formal description of reconfiguration rules in a graph transformation system. We begin by introducing a visual and graph-based language that itself results from a formal definition presented in [46] (section 3.1), and continues with the introduction of a transformation algorithm describing the transformation of this graph-based language to reference nets (section 3.2). The section will finish by introducing the formal reconfiguration of user interfaces (section 3.3).
3.1. Graph-based formal modeling of User Interfaces

VFILL is a visual and graph-based modeling language first introduced in [46]. It is based on a formal defined language called FILL that is convenient for formal transformation of the interaction logic modeled as FILL graph to reference nets. VFILL models the interaction-logic of a user interface, describing the user interface’s behavior as set of processes for processing input and output data. This kind of user interface modeling is based on a three-layered architecture, which is shown in Figure 1. Here, a user interface is separated into two layers: (a) its physical representation and (b) its interaction logic. The physical representation of a user interface is a set of interaction elements (IEs) for input like buttons or sliders, etc., or for output like text fields or more complex data visualization components. These IEs are connected to the user interface’s interaction logic, which models its behavior. The behavior of a user interface refers to the way data is transferred to or from the system, representing the third layer in the architecture. Thus, the data generated in the physical representation by such means as clicking events on buttons or inputting text in a text field is processed in an interaction process in the interaction logic and then sent to the system interface. The system interface is a well-defined set of system values that can be either read or changed from outside. Values that are generated by the system to be read from outside are processed in the interaction logic and sent to the physical presentation.

Figure 1: Three-layered architecture for modeling a user interface involving building mental models in a cooperative learning process.

In our current work, the system implements only elementary operations, like creating, de/encrypting a message, or sending it from one participant to another. The protocol that
describes in which sequence these operations should be executed and by whom (Alice, Bob, or TTP) will be modeled using reconfiguration techniques in the interaction logic of the user interface. The goal is that, while executing the protocol collaboratively, the students’ mental model changes while using reconfiguration techniques for user interface adaption until they have understood the protocol. A learning process implementing this approach will be part of investigation of the next section.

3.1.1. Type of Operation Nodes

Based on this architectural approach to modeling user interfaces, interaction logic can be formalized using VFILL as is defined in Figure 2. VFILL distinguishes three types of nodes: (a) operation nodes, representing system values (system operation), specific types of data processing units (interaction-logic operation), or connection elements to combine different interaction processes (channel operation); (b) proxy nodes representing connections to the physical representation and its IEs, and (c) BPMN nodes, which enable more complex modeling of interaction processes.

BPMN nodes are borrowed from BPMN modeling language for visual modeling of business processes. These nodes are able to fuse different interaction processes into one single process or to branch a single process to a set of downstream sub-processes.

Figure 2: Visual specification of VFILL, a visual language for modeling interaction logic and interactive user interfaces
Depending on their type (OR, AND, XOR), the outgoing edge in fusion cases is activated by the activation of the incoming edges. The same is true for branching. Depending on the type of node and an additional condition, one, a set of or all outgoing edges are activated. More information on the semantics of BPMN nodes in a VFILL graph can be found in Table 1.

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Branching</th>
<th>Fusion</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND</td>
<td>All outgoing edges are activated at the same time, in parallel. =&gt; parallelization</td>
<td>It waits for all incoming edges before activating the outgoing edge. =&gt; synchronization</td>
</tr>
<tr>
<td>OR</td>
<td>Depending on the conditions of the outgoing edges, one or more outgoing edges will be activated.</td>
<td>Depending on the given conditions, one or more edges will be synchronized.</td>
</tr>
<tr>
<td>XOR</td>
<td>Depending on the condition, exactly one outgoing edge will be activated.</td>
<td>Depending on the condition, it will wait for exactly one incoming edge before activating the outgoing one.</td>
</tr>
</tbody>
</table>

3.1.2. **Port and Proxy Nodes**

The connection to the physical representation of a user interface is modeled as proxy nodes. These nodes either send data from the physical representation to the VFILL graph (output proxy) or from the graph to the physical representation (input proxy). Proxies are connected to operations, and operations are connected to one another by data edges. A data edge is represented as a solid arrow between the proxy and the port of an operation or between the ports of two operations. Ports as well as proxies are typed; this means that only data objects of a certain type can be sent to a certain port or proxy. An example of a port is shown in Figure 2 in the lower part of the operation box.

3.1.3. **Channel operations**

For the connection of different interaction processes, channel operations can be used. Input channel operations represent the entrance of a channel, and output channel operations the exit. Channels themselves are represented by dashed arrows between channel operations.

The last node to be introduced is the consumer node. This node only consumes data objects from the process and thus defines the end of an interaction process in a VFILL graph.
3.1.4. Example of a VFILL model

Figure 3 shows an example of an interaction logic modeled in VFILL taken from [46]. The left interaction process models the data processing of a string which might, for example, result from an enter event of a text field on the physical layer. The entered string is sent to the interaction process as a data object at the moment the user depresses a certain key, generating an event with the content of the text field attached as a string object. This piece of data is then sent to an and-node, which splits the IP into three sub-processes sending the same string to one of them. The left sub-process generates an integer value, using an interaction-logic operation to transform the string. The result is passed to another interaction-logic operation, which multiplies the two incoming Integer objects of this interaction-logic operation. The other input value for this interaction-logic operation is passed from a system operation triggered by the string object in the rightmost sub-process. In this context, the string object’s only function is to trigger an operation. The result of the multiplication operation is then passed to a system operation, setting this value to the system value called B. The returned value from this operation is consumed by a terminator node.

The third sub-process triggers an input channel operation to send the string object to a second IP, shown on the right side of Figure 3. The process transforms the string coming from the left IP into another string and sends it to an interaction element for output. Another value is fetched from the system using a system operation. This value is sent to the or-node any time the ticker interaction-logic operation sends an Object to the system operation. This happens iteratively with a specified amount of waiting time between the objects sent. In this way, it is possible to acquire a current value from the system without an event being triggered by user interaction. The OR-node sends only the incoming data objects directly to its outgoing edge without any constraints.
Cooperative Reconfiguration of User Interfaces for Learning Cryptographic Algorithms

3.2. Transformation of VFILL to Reference Nets

Reference nets [24] are a special type of Petri nets offering various extensions to the basic Petri net formalism:

1. Token can be associated to complex object-oriented data types,
2. Transitions can be associated to each other by synchronous channels, and
3. Tokens can reference other net instances.

Point 3 in particular distinguishes reference nets from higher Petri nets offering synchronous channels like introduced by Jensen [16].

The main goal of extending higher Petri nets through a referencing mechanism is the possibility of simulating more than one net instance in parallel derived from one net pattern. This approach can be compared to object-oriented programming languages, where classes define patterns of objects (also called instances) during runtime. Thus, different net instances of one net pattern can have different markings but still have the same place/transition structure.

Another ability of reference nets derived from the reference mechanism and its close relationship to object-oriented programming languages is the calling mechanism to Java methods during simulation of the net. This mechanism is implemented mainly in context
of Renew [23], a simulation and modeling environment for reference nets. This ability is important for the transformation of VFILL graphs to reference nets and their later inclusion in the simulation of formal interaction logic and user interfaces.

Figure 4 shows the transformation of an interaction logic operation and a BPMN node. There are three main steps in transforming VFILL nodes into reference nets: (1) the representation of ports as places, (2) the representation of the operation itself as a transition calling a Java method, and after having finished the calculation, (3) returning to the net using either a second transition (left reference net in the upper part of Figure 4) or one transition that applies the calculation directly in the net through complex inscriptions.

The transformation of BPMN nodes is exemplarily shown for an AND node. Here, a transition is used to represent any incoming and outgoing edge as a place and as the node itself. This transition is labeled with the condition connected to the AND node in the VFILL graph. In the VFILL graph, it can be seen that the data should be sent to the outgoing edge (indicated in the condition by “a”). This is represented in the reference net by the variables labeling the edges. The outgoing edge has the variable a as part of the tuple. Thereby, the incoming value bound to a will be sent to the outgoing edge.

In all transformed sub-nets, edges are labeled using tuples. The first value represents a unique identifier that is an object-oriented representation of the original IE and a data object that is the second variable of the tuple representing the input data from the physical representation that should be sent to the system. The IE representation is important for later application of reconfiguration to the net described in the next section.

Figure 4: Examples of transformations of VFILL nodes to reference nets
3.3. **Formal Reconfiguration of Interaction Logic**

Formal reconfiguration of interaction logic is based on graph transformation systems. Ehrig et al. describe in [11] the transformation of Petri nets employing the double-pushout (DPO) approach from category theory.

In the DPO approach, a rule for transforming a graph $G$ to a graph $H$ is given in form of three graphs $L$, $I$ and $R$, as shown in Figure 5, as well as a matching function $m$. $L$ and $R$ are the left and right side of the rule, and $I$ is the interface graph. The difference between $L$ and $I$ describes what will be deleted in $G$, resulting in graph $C$, and the difference between $I$ and $R$ defines what will be added to $C$ to finally get $H$.

The example in Figure 5 shows this two-step transformation process based on the rules of DPO. The upper part of Figure 5 shows this approach with the nets $L$, $R$, and $I$. The matching function $m$ is implicitly defined here by numbering the places in the rule nets and the net to be transformed.

The difference between nets $I$ and $L$ consists in the edge between nodes 1 and 2. This means that this edge will be removed, resulting in net $C$. Next, the elements in net $I$ and net $R$—here the new node 3—will be added to net $C$, as will the edges between nodes 3 and 1 and between nodes 2 and 1, resulting in graph $H$.

![Figure 5: Example of a transformation rule for the double pushout approach](image)

In our work with J. Stückrath [41], this approach was extended to colored Petri nets. A tool was implemented to transform higher Petri nets along with inscriptions for transitions, places and edges. This implementation is based on a serialization of reference nets in PNML format [43], which is an XML-based language for representing Petri nets of different types.

The following section will describe the use of the formal framework for reconfiguration in the context of a cooperative learning process paired with an overview of the software we implemented and used.

4. **Cooperative Reconfiguration**
Cooperative Reconfiguration in the context of learning cryptographic protocols is the main topic in the current section, which begins with an overview of the cooperative learning process (section 4.1). Next, we introduce our software for modeling, reconfiguring and simulating user interfaces (section 4.2). The final section describes reconfiguration on the physical level paired with the reconfiguration of interaction logic (section 4.3).


Figure 6: Cooperative and iterative learning process using reconfiguration techniques for user interface for learning of cryptographic protocols

Figure 6 shows our approach to cooperative learning using reconfiguration techniques on a user interface that fosters the generation of a mental model of cryptographic protocols. In addition to a classroom-based introduction and a final analysis and conclusion, the central aspect of this process is the interface reconfiguration phase as an iterative subprocess. Here, the goal for the students is to build the protocol step by step using an interface to which they apply reconfiguration operations.

The initial interface is shown in Figure 7, extended by additional descriptions of the elements that are only presented to the students during the introduction phase. The figure shows three IEs that represent the three participants in the protocol: Alice (left), Bob (right), and the TTP (in the middle). Any IE that represents a participant is split into three areas: (a) the knowledge area, representing the data or information the participant has in a certain situation of the protocol (e.g., keys), (b) the mailbox, containing a message to be sent by the participant or a message received from another participant, and (c) the toolbox, representing the area of operations the participant can apply to the mailbox content. In the lower part of the interface, a set of operations shown as buttons labeled
with icons that indicate their functionality, or concept. Figure 8 shows all the operations in detail and describes their functionality in the protocol.

Figure 7: Initial user interface for an iterative learning process involving the reconfiguration of user interfaces

Two major groups of operations are necessary for the protocol: (a) the de/encrypt buttons with an icon representing the key that will be used to de/encrypt a message in the mailbox and (b) the creation of message buttons that generate the message indicated as an icon on the button and add it to the mailbox. Two further operations are also shown: (c) the creation of a session key AB and (d) the selection of keys from a key data bank (KDB) after Alice has sent the initial message containing the information she wants to communicate to Bob.
This initial user interface has to be reconfigured using the following reconfiguration operations to model the protocol:

1. Duplication: This reconfiguration duplicates an operation. For example, the de/encrypt operation with Alice’s key is used by Alice and by the TTP.

2. Deletion: This reconfiguration deletes an operation.

3. Toolbox association: This reconfiguration associates an operation to a toolbox. With this reconfiguration, operations are connected to the roles ‘Alice’, ‘Bob’ and ‘TTP’.

4. Communication connection: This reconfiguration combines participants via a communication channel. It can be used to define between which participants a message can be sent and who is the sender and/or the receiver. This operation also adds a send button to the small upper box in the mailbox.

5. Sequencing: Operations that are already associated to a participant’s IE can be combined in a new operation associated to a new button in the toolbox. This new button executes the combined operations in a specific sequence.

The resulting changes on the physical layer are shown exemplarily in Figure 9 for the association, the communication connection, and the sequencing reconfiguration. The association of an operation like the de/encrypt operation using Alice’s key is shown on the left. The operation button is included in the box representing her toolbox. If the reconfiguration creates a communication connection, the two mailboxes are connected by an arrow and a button is allocated to the participant who is the sender in the new connection (here Alice), labeled “send TTP”. By pressing the button, the content of Alice’s mailbox is sent to the TTP. The rightmost example in Figure 9 shows the results of a sequencing reconfiguration applied to message creation and a de/encrypt operation. The result is a new button with two icons in the order they are executed by pressing it.
Cooperative Reconfiguration of User Interfaces for Learning Cryptographic Algorithms

All reconfiguration operations, including the deletion and duplication operations applied to the user interface, influence the interaction logic mainly by applying graph transformation rules to the reference net representing it. In this way, for every reconfiguration operation, a transformation pattern has been implemented that is instantiated using the information offered by the user. He has to state to which IE and to which part of the interaction logic a certain reconfiguration operation should be applied by clicking on it or by other means. Figure 10 shows an example of two IEs and their associated part of the interaction logic as a reference net as well as a new IE generated through sequencing the two single operations. The original nets that are associated to the single operations are indicated in Figure 10 as gray boxes and circles and as thin arrows. The new structures introduced by the reconfiguration are shown as black transition boxes and thick arrows. The corresponding DPO rule is shown in Figure 11, where the matching function \( m \) is symbolized by the numbers of the places in the rule and by the numbers in the reference net (identical numbers indicate a match).

---

**Figure 9:** Examples of reconfiguration operations applied to the initial user interface to simulate the NSP

**Figure 10:** Examples of an applied sequencing reconfiguration and its effect on the underlying interaction logic given as a reference net
Before describing the transformation rule and its application to the interaction logic in detail, let us turn our attention to the net itself, first without involving the extension through reconfiguration. Two processes can be seen (de/encrypt message and create message), each connected to a button. In both cases, the IE is associated to a transition in the net labeled by the method buttonPressed that sends an Event object to the net as a token. The colon indicates that these transitions can be fired from outside the net. In the later simulation environment, this will be handled by a Java source code. After firing the transition of the message creation operation from outside (by clicking the button), the process for the creation of a message m is started by creating a tuple ['ie1', e] where 'ie1' is an identifier of the message creation IE and e is an object representing the event. This tuple-token is then assigned to the place indicated by ‘3’. Now, the downstream transition can fire, creating a message by calling the function createMessage and binding this to the variable m. The message is then assigned to the downstream place ‘2’ as a tuple paired with a variable that in this case is bound to the value ‘ie1’. Now the consuming transition to the right of place ‘2’ can fire because the variable vie is bound to ‘ie1’.

The processes for the de/encrypt operation are more or less the same, differing only at place ‘1’. Here, a message m has to be offered by the upper leftmost place. This message can be obtained from the mailbox if the de/encrypt operation is associated to a toolbox using the association reconfiguration. This reconfiguration has to connect this specific “message” place to the subnet of the IE representing the mailbox.

Now, a sequencing reconfiguration should be applied to the net such that the resulting net executes the two subnets as sequence. The rule shown in Figure 11 does not delete anything (there is no difference between L and I) but adds two transitions and three edges to the initial reference net (see R). These new elements are shown as black transitions and thick arrows in Figure 11. If the newly generated button is pressed, the event object is first sent to the sub-process for the creation of a message and then redirected by the indicator ‘ie3’ to the sub-process for de/encrypting a message. Still, the resulting message will only be consumed and nothing will happen, but applying a reconfiguration for operation association first to the initial operations would redirect the resulting message to the subnet representing the mailbox of the participant IE.

Figure 11: DPO rule for the applied transformation of interaction logic shown in Figure 10
This creation step in the process is followed by a verification step, which compares the resulting net to an expert model. This can be implemented using state space analysis to prevent the use of graph morphisms, etc. Another option is to simulate the protocol created by the students and match the resulting markings to a model pre-assembled by an expert.

The resulting differences are reported to the students so that they are able to apply further reconfigurations to the user interface. The result of the whole process is a model of the NSP as reference net.

The construction of the protocol using formal reconfiguration techniques can be applied cooperatively to the user interface in two ways: (a) All students work at one workstation and one interface, or (b) every student works on only one role using the initial interface (Alice, Bob, or TTP). It would also be suitable to combine both approaches based on the described framework: First, the students work together to create the protocol, and then they work individually, with each creating only one part of the protocol. Students can then create the roles one after the other, always paired with the verification mechanism that provides a response from the system.

With distributed learning, verification is also manageable using the formal approach described. Here, the reference net is modeled using three reference nets—one net per role. A fourth net, reconfigured by all participants, models the communication between the participants, thus modeling the exchange of messages and thereby the exchange of tokens between the three nets. Using state space analysis on this reference net construction should yield the same result as one net describing the whole protocol without separation into the different roles.

The learning process concludes with a simulation of the created protocol using the resulting user interface and a classroom-based discussion. This simulation can be exported to CoBo or simulated in the UIEditor, a tool for modeling, simulation, and reconfiguration of user interfaces presented in the next section.

4.2. **UIEditor-Software for Formal Modeling and Simulation of User Interfaces**

The UIEditor is a tool for creating, simulating, and reconfiguring user interfaces, first presented in [44]. It has three parts: (a) an interactive editor for creating the physical representation and interaction logic based on the interactive modeling of VFILL graphs, (b) a simulator that combines the simulator Renew for simulation of reference nets with the simulator for the system and the physical representation of the user interfaces, and (c) the editor for reconfiguring a user interface. Modules for transforming VFILL graphs into reference nets and a module for applying DPO rules to a reference net based on XML serialization have also been implemented [41].

The creation editor is separated into two windows: (i) the editor for the physical representation and (ii) the editor for interaction logic. Both editors are shown in Figure 12: On the left side, the editor for interactive modeling of VFILL graphs is shown; on the right is the editor for creating the physical representation of a user interface. Basically, both editors implement the drag-and-drop metaphor for adding and deleting operations.
and BPMN nodes for modeling a VFILL graph together with interaction elements for the physical representation. Edges between proxies and ports of operations are added by point, click and drag operations as known from other editors for modeling graph-based structures. Toolbars in both editors offer additional functionality, such as standard operations for loading or saving modeled interfaces. In VFILL modeling, the toolbar mainly provides operations for adding channel operations or deleting selected nodes.

The creation of an interaction process for a specific IE starts by clicking on the IE resulting in the display of the associated interaction process in the VFILL editor. All proxies that represent the events emitted by the IE are shown as are the values that can be submitted to it. For the definition of channels between the various interaction processes of different IEs, channel operations have to be added to the interaction process and channels have to be defined using a context dialog that shows up after double-clicking the channel operation. Guard conditions for BPMN nodes are added in the same way by double clicking an added BPMN and adding the guard condition in the presented dialog box.

To use the new user interface in a simulation, first a simulation file has to be created by changing the mode of the UIEditor to simulation mode and then selecting the option ‘Create Simulation File’ from the simulation menu. After loading the created file, the simulation can be started by pressing the start button in the toolbar. The simulation module takes cares of connecting the reference net on the one hand to the physical representation and to the system on the other. Renew simulates the reference net, and its token plays and handles transition calls to and from the reference net.
For reconfiguration, any running simulation has to be stopped and the UIEditor switched into reconfiguration mode, which looks much like the simulation window. A toolbar offering all activated reconfiguration operations is displayed.

After the user has finished reconfiguring the user interface, the simulation can be restarted. The UIEditor loads the new interface, the new reference net, the physical representation, and the system, and starts all components.

5. Evaluation

The entire approach of using reconfiguration techniques for user interface adaptation is encouraged by former cases studies of cooperative creation of user interfaces for distributed simulation of cryptographic protocols [45]. This evaluation was conducted in the year 2009 at the University of Duisburg-Essen and described in detail in [48]. In the case study 66 students participated, verifying the hypothesis that cooperative creation of keyboards supports the understanding of cryptographic protocols and enables complex problems to be analyzed and solved. This hypothesis was tested by having participants complete a questionnaire. Additionally, a post-test was conducted. The evaluation of the questionnaire showed a highly significant difference between the cooperative and non-cooperative groups in a two sample t-test ($t = 7.1$, $p < 1\%$) evaluated using SPSS© tools in version 17 (PASW Statistics 17©, SPSS Inc.), where the cooperative group rated their individual estimated learning progress ($M = 3.68$) higher than the non-cooperative group ($M = 2.82$) on a 5 point Likert scale. The post-test showed a high enhancement of knowledge by nearly 1 point on a [1, 5] scale, but no significant difference between the cooperative and non-cooperative groups ($t = 0.33$, $p = 74.3$ %). Still, the perceived learning success is higher in a cooperative learning scenario. A reason that both groups behave similarly in the post-test may be the type of questions in the test, which evaluated the extent to which the participants had been able to transfer the learned knowledge to a more complex scenario by assuming the presence of an intruder in the protocol. To show a difference in learning success between the two groups, the post-test must put greater emphasis on how to evaluate knowledge about the protocol in the standard scenario without an intruder because that was the content learned by the students.

The impact of reconfiguring user interfaces to the mental models of human users was shown in a case study conducted in June 2010. This case study will be covered in future publications. However, good results were found regarding error reduction in failure handling of complex systems.

6. Conclusions and Future Work

We have presented a new approach to the learning of cryptographic protocols based on a new and formal approach to modeling and reconfiguring user interfaces. The visual language VFILL was introduced as new modeling approach for describing interaction logic. Interaction logic itself describes the behavior of a user interface, which is nothing other than a set of elements for processing data emitted by the physical representation of the interface (events triggered by pushing buttons, etc.) or by the system to be controlled.
by the user interface (changes in system states, etc.). By transforming VFILL graphs to
reference nets, a special type of Petri nets, VFILL acquired a formal definition of
semantics and a framework for formal reconfiguration. In cooperation with J. Stückrath
[41], we implemented a framework for transforming higher Petri nets based on the formal
approach described by Ehrig et al. [11]. The formal transformation and reconfiguration
framework for VFILL and reference nets has been integrated into an interactive modeling
environment called UIEditor for visual modeling, simulation and reconfiguration of user
interfaces.

With the support of this tool and motivated by the results of various case studies
presented in the former section, a new iterative learning process paired with a user
interface implementation for a cryptographic protocol (the Needham Schroeder key
exchange protocol) was introduced and described. This process is mainly an iteration of
collaborative modeling of the protocol using reconfiguration techniques on an initial user
interface following by a verification and simulation step using formal methods for state
space analysis and a visual simulation environment of cryptographic algorithms like
CoBo [21].

An important aspect of future work is the evaluation of the described learning
process. Here, the main hypothesis—that the use of reconfiguration in a cooperative
learning environment will improve learning success relative to its use in a non-
cooperative environment—has to be further investigated. The general impact of
reconfiguration on building mental models and on investigating the influence of
reconfiguring user interfaces on learning processes should be the subject of further case
studies. This research model could be extended by introducing an intruder to the scenario
the students have to deal with. The resulting protocol generated by reconfiguration can be
verified by formal approaches like that described in [22].

This paper is based on a publication in 2009 in CRIWG in Peso da Regua, Portugal
[47].

References
694X(79)90023-1.
2. M. Azmitia, Peer Interaction and Problem Solving: When Are Two Heads Better Than One?,
3. N. Baloian, and W. Luther, Cooperative visualization of cryptographic protocols using concept
4. N. Baloian, H. Breuer, and W. Luther, Concept keyboards in the animation of standard
5. N. Baloian, and W. Luther, Algorithm Explanation Using Multimodal Interfaces, *Proc. of the
25th international conference of the Chilean Computer Science Society*, Los Alamitos, 2005,
pp. 21.
6. B. Baumgarten, *Petri-Netze: Grundlagen und Anwendungen* (Spektrum Verlag, Heidelberg,
1996).
426.