Electronic Communications of the EASST
Volume 30 (2010)

International Colloquium on Graph and Model Transformation On the occasion of the 65th birthday of Hartmut Ehrig (GraMoT 2010)

Formal Modeling of Communication Platforms using Reconfigurable Algebraic High-Level Nets

Tony Modica, Kathrin Hoffmann

24 pages

Guest Editors: Claudia Ermel, Hartmut Ehrig, Fernando Orejas, Gabriele Taentzer
Managing Editors: Tiziana Margaria, Julia Padberg, Gabriele Taentzer
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Formal Modeling of Communication Platforms using Reconfigurable Algebraic High-Level Nets*

Tony Modica[1], **Kathrin Hoffmann**[2]

Technische Universität Berlin[1]
Hochschule für Angewandte Wissenschaften Hamburg[2]

Abstract: Communication nowadays is based on communication platforms like Skype, Facebook, or SecondLife. The formal modeling and analysis of communication platforms poses considerable challenges, namely highly dynamic structures and complex behavior. Since most of the well-known formal modeling approaches are adequate only for specific aspects of communication platforms, in this paper we introduce the approach of reconfigurable algebraic high-level nets with individual tokens and show in our case study Skype that this approach is adequate for modeling the main aspects and features of communication platforms.

Keywords: Algebraic high-level nets, higher-order nets, individual token approach, communication platforms, Skype

1 Introduction

During the last decade, mobile and adaptive communication systems like Skype, Facebook, or SecondLife have become more and more important. These systems have several aspects in common. In mobile and adaptive communication-based systems, communicating entities (actors) can transmit content, which is contextually interpreted. Actors may join, move in or leave communication platforms, where the actors' preferences, access rights and roles are respected and define a temporary set of communicating partners and a context of interpretation for communicated data.[1]

It is desirable to have a formal modeling technique for communication platforms, so that we can specify the features of such systems in an appropriate way and are able to simulate, test, and analyze/verify them by using not only the structure but also the formal semantics of the modeling technique. General properties of interest are related to consistency, safety and security requirements, liveness, termination etc. We have observed that most of the well-known modeling techniques like UML and actor systems [Agh85] or formal specification techniques like process algebras [Mil99], low-level and high-level Petri nets [Rei85, JR91], algebraic specification [EM85] and graph transformation [EEPT06], as well as different kinds of logic are

* This work has been partly funded by the research project forMA$_l$NET (see tfs.cs.tu-berlin.de/formalnet) of the German Research Council and by the Integrated Graduate Program on Human-Centric Communication at Technische Universität Berlin

[1] The notion of Communication Spaces has been coined in the research area "Modeling and Engineering of Computer Supported Communication Spaces" of the recently founded Innovation Center Human-Centric Communication at Technische Universität Berlin as an characterizing concept of communication systems with these properties and as an ontology for describing mobile and adaptive communication systems.

able to model and/or analyze these aspects only partially. Conventional modeling techniques for communication-based systems like classical Petri nets and the UML are restricted to model static, immutable network topologies. Graph transformation systems in contrast are dynamic in their structure but lack a notion of behavior. Of course, appropriate graph transformation systems may also be used to simulate e.g. the behavior of Petri nets but it seems advisable to distinguish user behavior from reconfiguration and to possibly use standard results for the behavior analysis of Petri nets. Moreover, we believe that diagrammatic models like Petri nets and graphs supporting visual modeling and visual behavior simulation can have advantages for system modeling w.r.t. readability and understandability, though there is no standard measure for these properties.

In the context of concurrent and distributed systems Petri nets, first introduced by C.A. Petri in [Pet62], are a well-known and widely used formalism and have been employed in practical applications in many different areas (see e.g. [Rei85, RT86, MM89, MM90, Win87, Bau90]). Their graphical representation and formal semantics excellently support the modeling, simulation, and formal analysis of such systems. High-level net classes are obtained by combining Petri nets with an appropriate data type part. Most prominent are coloured Petri nets [Jen92, Jen94, Jen97], a combination of Petri nets and a high-level programming language, which is an extension of the functional programming language Standard ML. Coloured Petri nets offer formal verification methods and an excellent tool support, which has been used in numerous case studies within a large variety of different application areas. Apart from this, there are algebraic high-level (AHL) nets [EPR93, EHP+02], which give rise to a formal and well-defined description due to their integration of classical algebraic specifications [EM85] into Petri nets.

Especially in communication platforms we have to deal with highly dynamic structures and behavior; most notably, the number of users known to the system can grow or decrease during runtime. In order to maintain a variable number of users, we also need a possibility to reconfigure the structure. So we advocate the integrated formal modeling technique of reconfigurable AHL nets for communication platforms like Skype, where on the one hand the data type part represents users identities and their communication data and on the other hand suitable rules express the essential features of communication-based activities. Moreover, we need to change the marking of a net freely by appropriate rules to be able to process data in a distributed way in contrast to the local effect of transition firing. Thus, to achieve an adequate dynamic reconfiguration at runtime, not only the structure of an AHL net is manipulated by rule application but also its marking. A further essential aspect of communication platforms that needs marking-changing rules is multicasting, i.e. transmission of data to selected actors, which can not be realized adequately by the classical firing behavior of Petri nets [BEE+09].

This paper is organized as follows: In Section 2, we show how to model the main requirements of communication platforms in an adequate way with reconfigurable algebraic high-level nets with individual tokens by our case study Skype as a typical example of a communication platform. Section 3 gives an overview of the analysis results for AHL nets with marking-changing rules. Section 4 discusses higher-order markings as concept to provide a control structure for more complex systems to model. Section 5 ends with the conclusions and an outlook to future work.

2 Formal Modeling of Skype with Reconfigurable AHOI Nets

Skype is currently a widely used Internet telephone software and can be obtained and used free of charge. In its basic version it features most of the functions a communication tool is expected to have like multi-user conferences and contact management and therefore Skype is a reasonable choice for a modeling case study of communication platforms. In this section, we discuss the principles we chose to model Skype with AHLI nets and demonstrate how we model several aspects of Skype features according to these principles.

Contact Management A user should call or chat only with users who explicitly allowed this specific user to contact them. In many communication applications like Skype, this is solved with contact lists. To enable a user to add them to his contact list, Skype offers a function to search for contacts in a white pages directory of all registered users. The asked user can deny this request.

Calls A user can directly call one user on his contact list like in a telephone call. The callee gets notified by its Skype client and he may then accept the call unless he is already in another call or conference. In this case he has to end the current call before accepting the incoming one. This means that calls and conferences are exclusive communications and a user can only participate in one at a time.

Conferences A conference is a generalized form of direct call where several participants (up to a technically bounded number) talk to all other participants. A direct call can be considered as special case of a conference with two participants. In Skype, if someone starts a conference he is designated as its host who is the only one who can invite other people and kick them out after having joined the conference. If the host quits a conference, it is terminated, i.e. the other conference participant can not send any further messages afterwards.

Call Forwarding Known from service hotlines in the field of commercial communications, call forwarding means that the callee transfers an incoming call to another contact in his list as if the caller would have called the contact he is being forwarded to directly, even if the caller does not know this contact. In Skype, call forwarding only works for direct calls and is no longer available once a direct call is expanded by the caller to a conference of at least three people.

Chats Chats are similar to conferences but in contrast they are mostly based on sending messages that do not have to be perceived immediately like acoustic messages in calls, wherefore we classify chats as non-exclusive communication. In Skype, a user can participate in as many chats as he likes in parallel, add references to them in his contact lists, and look through each chat's history when it is opened[2]. Apparently, it is not possible to delete the history of a chat in the Skype client, so we simply consider Skype chats as persistent and users can not actively leave

[2] However, management of chats in the Skype client is not clearly explained; you may select a set of contacts and start a chat and invite and kick contacts like in conferences. But if you already have created a chat for a set of contacts the old chat and its history are displayed when selecting this particular set of contacts again.

a chat they have joined before but only be invited and kicked by other participating users. This is another difference to conferences, where only the host has the right to invite and kick other users; for this reason, a chat does not have an designated host.

In the rest of this section, we discuss how to use AHL nets with marking-changing rules to realize these main Skype features.

2.1 Skype Clients as AHLI nets

AHL nets as a fundamental, visual and formal model in concurrency, have recently been subject for suitable extensions. A useful approach for transformation of marked AHL nets is the rule-based approach with double pushouts of [PER95], which has been proven to have the properties of $\mathscr{M}$-adhesive systems (also known as weak adhesive high-level replacement systems) in [EEPT06] and, thus, providing many analysis results concerning the local Church-Rosser property, parallelism, and concurrency of transformations. In [PER95], the marking of an AHL net has been defined as an element of the commutative monoid $(A \otimes P)^{\oplus}$, i.e. a sum of pairs (a,p) where p is a place in the net and a is a data element of the algebra's carrier set $A_{type(p)}$ that resides as a token on place p. Unfortunately, in $\mathscr{M}$-adhesive systems for nets with such "collective" markings, we can only define rules that must not change the markings of places[3], which is too restrictive for our modeling approach.

AHLI nets For this reason, we use the formal technique of AHL nets with "individual" tokens (AHLI nets) because for AHLI nets we can overcome this restriction and formulate $\mathscr{M}$-adhesive systems with rules that arbitrarily alter the marking of places [MGE+10]. Thus, essential features like multicasting can be modeled in an appropriate way, which we show in a case study that unites and extends the work done in [BEE+09, Mod10, MEE+10].

An individual marking for a net is a pair (I,m) with I the set of individual tokens and $m : I \to A \otimes P$ is the marking function, assigning the individual tokens to the data elements on the places. Each AHLI net with a placewise finite individual token marking (I,m) can be interpreted as an AHL net with marking $\sum_{i \in I} m(i)$.

As a main principle, we strictly distinguish user behavior from system reactions, i.e. we represent everything a user can do like entering data or pushing some buttons in its Skype client by some transition in an AHLI net. Everything else that happens as a consequence to user behavior, especially the extension or restriction of possible user actions, is realized by reconfiguring rule applications. In the following, the whole Skype system is always represented by a single AHLI net, possibly with discrete components that represent idle user clients or other structures like ongoing conferences. We discuss an extension of AHLI nets to further structure and control the components of a Skype system is introduced in Sect. 4, but as a first step we model Skype in a single AHLI net.

[3] In fact, to simulate the change of a place's marking this place could be deleted and recreated by the rule with a new marking but this is not possible in every context due to other structural restrictions.

Data Types for Skype AHLI Nets For the Skype case study, we fix a common signature **Skype**$-\Sigma$ for all nets describing a Skype system and for all nets in the transformation rules. This signature consists of the following sorts and operations. We also give the carrier sets of the **Skype**$-\Sigma$-algebra A we are using for the following Skype nets.

Bool are the usual truth values $\{True, False\}$ for expressing the state of condition predicates.

SkypeName is the type of words in $\{A,\ldots,Z,a\ldots,z,0,\ldots,9\}^*$ for identifiying the clients' user names.

Data is the type of words in $\{A,\ldots,Z,a\ldots,z,0,\ldots,9\}^*$ for the data to be transmitted in calls and chats. We chose it for this example but it may be any kind of data.

ComMode is the finite type $\{Call, Chat\}$ describing the possible kinds of communications in Skype.

ComRequest is a special type whose elements describe the requests for a communication with some user. There is a simple constructor operation $req : SkypeName \times ComMode \to ComRequest$ that builds a request for a name and a mode.

State elements from $\{Online, Offline, DND\}$ describe the state of a Skype client. States can be compared with a predicate $notEqual : State \times State \to Bool$.

Log is a type for logging communication events. There is a constant *Empty* for the initial log and two constructor operations $sent : Log \times Data \to Log$ and $rec : Log \times SkypeName \times Data \to Log$ to add events for sent data and received data (with information about the sender) to a log.

Control is a singleton type $\{\bullet\}$ that is used for controlling the firing of transitions, e.g. to ensure that a transition can be fired only once.

The carrier sets in the **Skype**$-\Sigma$ algebra A for the sorts *ComRequest* and *Log* consist only of the terms over the given constructor operations. Note that this algebra is used for concrete Skype nets only and we describe special algebras for the nets in transformation rules in Subsect. 2.2.

Graphical Notation For the nets and rules we use a simplified graphical notation of AHLI nets as in Fig. 1. The rectangles are transitions and the ovals are places with inscriptions denoting the name and type of a place, e.g. in the lower left the place *User* of type *SkypeName*. Below the place's name we arrange the values of the tokens that mark this place. For example, on place *User* lies one token with the algebraic value *Alice*. Though we are dealing with individual token markings, the actual individuals are not relevant, so we omit the token's individuals in the graphical notation and give only the values on the corresponding places.

The arcs are inscripted with **Skype**$-\Sigma$ terms over variables. A special short notation are inscriptions like $s! = Offline$ on the arcs going out from the place *State*, which means formally that the arc is inscripted with s only and the adjacent transition has a firing condition $notEqual(s, Offline)$, while the firing condition sets of the other transitions are empty.

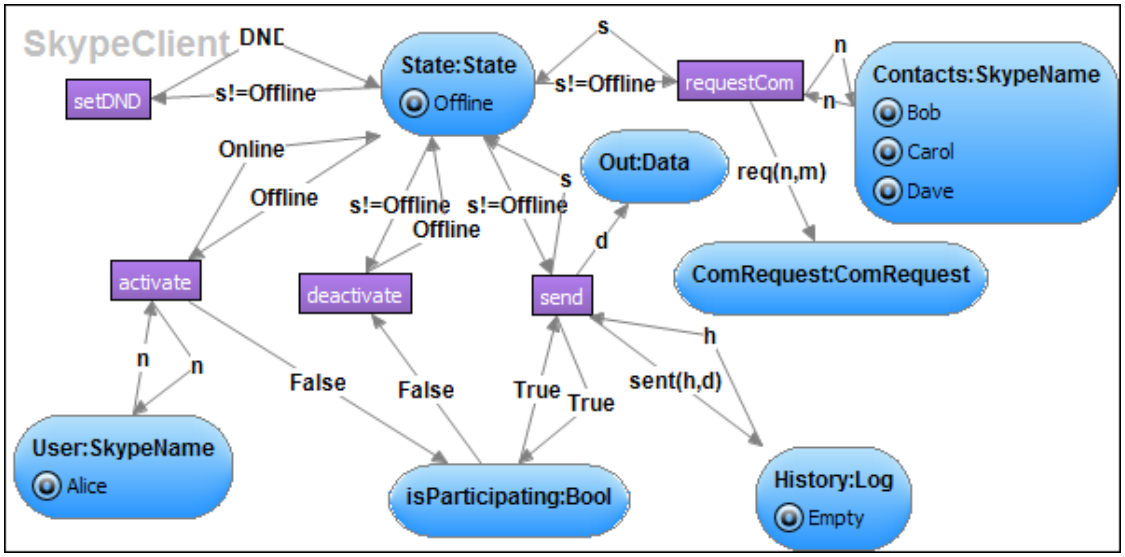


Figure 1: Net component for a single Skype client

Client Net Components The net component in Fig. 1 represents the Skype client of the user Alice. We assume that each registered user is represented by such a client in the overall Skype AHLI net, though they are unconnected (yet). The owner of a client is determined by the *SkypeName* token on the place named *User*, i.e. this client belongs to a user *Alice*. Moreover, it is currently in the state *Offline* as one can tell from the corresponding place. In this configuration, the only activated transition is *activate*, which on firing replaces the *Offline* state with *Online* and adds a *False* token to place *isParticipating*.

In this basic form, a client can change its state between offline, online, and "do not disturb", request a communication to a known user on its contact list, and send data if it is participating in a conference. In the following, we show an example firing step for transition *activate* in Fig. 1.

Example Firing Step Intuitively, activating the client should result in a token value *Online* on place *State*. Formally, we first have to check that *activate* is enabled, i.e. that for each term on its incoming arcs there is a corresponding token on the place the arc is going out from. As arcs are inscripted with terms over variables (or here just with variables), we need a variable assignment $asg : Var(activate) \to A$ assigning algebra values to the variables ocurring on the arcs adjacent to *activate*. If there is the right number of tokens on the corresponding places with the same values as the evaluated terms (with asg) we say that *activate* is activated under the assigment asg.

In the marked AHLI net Fig. 1, we see that place *State* carries a token with value *Offline* which is demanded by the arc going to *activate*. Further, for the variable assignment asg with $asg(n) = Alice$ we have obviously that the evaluation of n is the same value as of the token on place *User*. These are all arcs incoming to *activate*, so *activate* is actually activated under this asg and can fire.

When firing, *activate* consumes the aforementioned tokens and as a result produces tokens

with the values of the evaluated terms of its outgoing arcs on the corresponding places. In the case of Fig. 1 and $asg(n) = Alice$, *activate* produces one token with value *Online* on place *State*, one with *False* on *isParticipating*, and another with $asg(n) = Alice$ on *User* as depicted in Fig. 2.

The purpose of the other transitions is discussed briefly in the following sections of the corresponding Skype features.

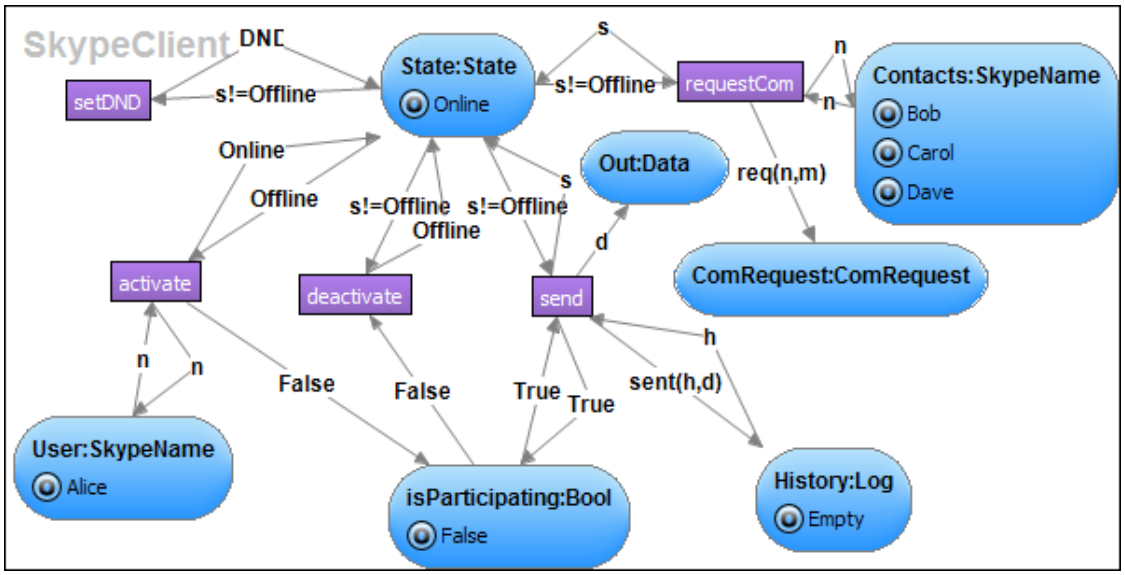


Figure 2: Activated net component for a single Skype client

2.2 Reconfiguration of Skype Clients

Following the guideline of behavior distinction, we allow a client just to announce a request for a call to a known client (i.e. whose owner name is present on its place *Contacts*) by firing *requestCom*. For example, the client in Fig. 2 can fire *requestCom* with variable assignment $asg(n) = Bob$, $asg(m) = Call$, if its user wants to give a call to the client of user *Bob*, which produces a token value $(Bob, Call)$ on the callee client's place *ComRequest*. This request enables the system to execute it with the application of the rules that we discuss in the following in more detail. We assume that the Skype system promptly reacts to requests with application of the rules that are currently applicable.

Reconfiguration of AHLI nets Next, we informally describe a special form of AHLI transformations that is sufficient for modeling and understanding the case study in the following section.

Formally, we use AHLI transformation rules following the graph transformation-based approach with double pushouts [EEPT06]. Rules are spans of morphisms $\rho : L \leftarrow K \rightarrow R$ with a left-hand side L, a right-hand side R, and an interface K being the intersection of L and R[4], all

[4] This means that the rule morphisms are inclusions.

these being AHLI nets with the same fixed signature **Skype**$-\Sigma$ and a special **Skype**$-\Sigma$-algebra for this rule. For a rule ρ, we first choose a family $Y_\rho = (Y_{\rho,s})_{s\in S}$ of variables that we want to use as variable tokens on the rule net places, such that Y_ρ is disjoint to the variables on the net's arcs. We then define the algebra for rule ρ as $A_\rho = T_{OP}(Y_\rho)$, i.e. the **Skype**$-\Sigma$-term algebra over the variables of Y_ρ. Due to space limitation in this paper a rule is depicted by $L \to R$, while the interface is obvious.

To apply a rule $\rho : L \leftarrow K \to R$ to an AHLI net AN, we need a match morphism $o : L \to AN$ describing the occurrence of L in AN. An occurrence match morphism is sufficiently specified by mapping the places, transitions, and tokens of the left-hand side into the target net AN. For mapping the tokens, a match needs a token variable assignment $asg : Y_\rho \to A_{AN}$. A rule ρ is applicable at a match if the following gluing conditions (cf. [MGE+10]) are met w.r.t the match: (1) if a place p in AN is adjacent to a transition t that is not in the match then p must be preserved by ρ, and (2) if a token in AN that is not matched lies on a place p, then this token must be preserved by ρ.

In this case, we achieve the transformation step $AN \overset{\rho,o}{\Longrightarrow} AN'$, where AN' results from deleting all parts in AN that are matched by parts in L that do not occur in R and then creating the parts occurring only in R and not in L. We give a detailed example of a rule application for the rule in the next paragraph.

Initiating a Conference We do not distinguish one-to-one calls from conferences but we use calls in the sense of a special initial case of conferences. Because of this, we go directly for conferences and do not treat calls explicitly. We refer to Subsect. 2.3 for how to model data transmission and focus on the conference management in this subsection.

Fig. 3 shows the left-hand and right-hand side of the rule ExecuteCallRequest1, which executes a call request by connecting the client components that play the role of the host (who requested the call) and the *callee* with a newly created conference structure. Given a request for a call as a token *req(n,Call)* (with some *SkypeName* value *n*) on the *ComRequest* place of any client component, the rule ExecuteCallRequest1 in Fig. 3 can be applied. This rule application extends the host's behavior options with one transition *quit* and the callee's ones with three transitions *join,refuse,leave*.

In Fig. 4(a), an example Skype AHLI net with two unconnected client components is depicted. Their owners are *Alice* for the left and *Bob* for the right client and both clients are in the state *Online*. In the following, the token variables of a rule are just the token terms that are not already defined in **Skype**$-\Sigma$ (u_1, u_2 in ExecuteCallRequest1) and their type can be deduced by the corresponding place and operation types the variables are occurring within.

The token variable assignment $asg : Y_{ExecuteCallRequest1} \to A$ with $asg(u1) = Alice, asg(u2) = Bob$ and the place matching indicated by similar place names, e.g. $UserA \mapsto User1$ etc. (also denoted with the dashed boxes in Fig. 4(a)), describes a valid match for the rule ExecuteCallRequest1.

The result of applying the rule on this match is shown in Fig. 4(b): The newly created nodes and transitions are highlighted with green boxes and the changes in the marking (deletion of the request token and replacing *False* with *True* on *isParticipatingA*) is highlighted with red circles.

Extended Client Behavior In the reconfigured Skype AHLI net, the host may *quit*[5], which sets a request to disband the conference. The *Conferencing* place holds the names of all users that actively participate in this conference. We consider a client as participating if it can send and receive messages from all other participants, i.e. it has been invited and joined the conference but has not left it yet. At the beginning of a conference, the host is the participating client. Initially, if the callee is not already participating in another conference he has the choice to either *join*, which would copy his name to the *Conferencing* place, or to *refuse*, which signals a request to disconnect the client from this conferencing structure. Note that the firings of *join* and *refuse* are mutually exclusive due to the *Invited* place type *Control*, which carries a black token to allow firing either *join* or *refuse*. After having joined, a client may *leave* which removes the client's name from the *Conferencing* place and announces the request to disconnect like transition *refuse* would do. Finally, the callee can choose to forward the call to another of its known clients (not the host) by firing *requestForward*, which produces a request token *req(n,Forward)* on the place *ForwardRequest*.

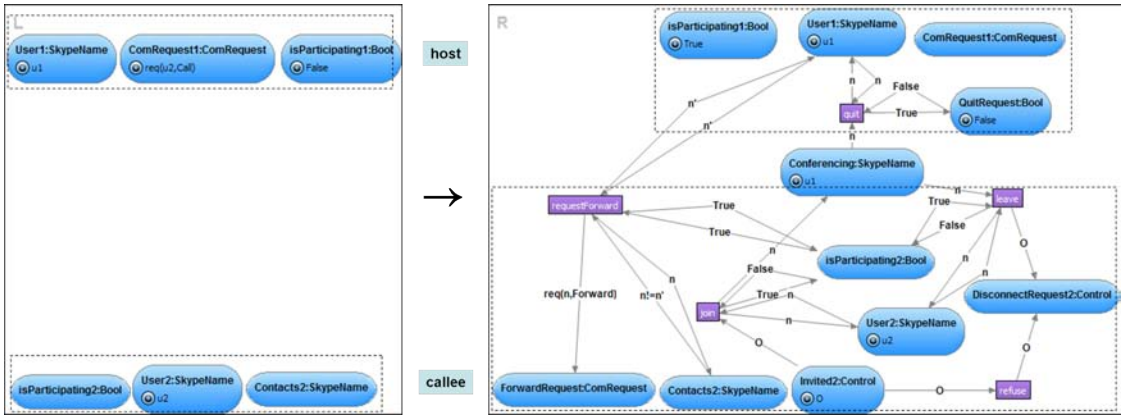
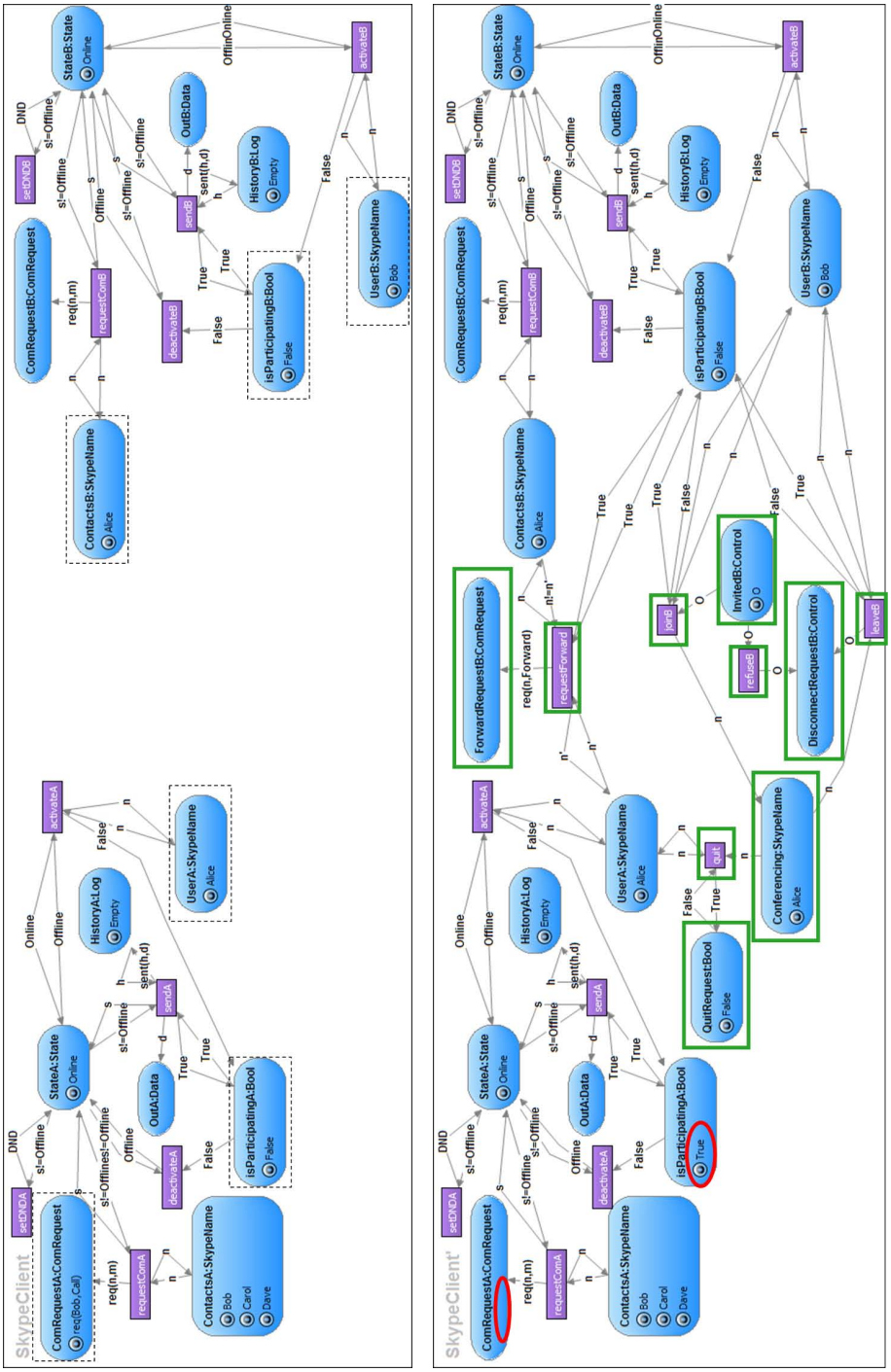


Figure 3: Rule ExecuteCallRequest1

Ensuring sensible Rule Applications Of course, rules that execute requests by reconfiguring the AHLI net must not be applied to random places of the appropriate type somewhere in the whole system but to specific places of a client in order to yield sensible structures. For example, in the left-hand side of the rule in Fig. 3, we assume the upper three places to be matched on the corresponding places of one client component. We can restrict applications of this rule on desired matches by demanding that the matches satisfy (positive) application conditions in the sense of [EEPT06].

Formally, an application condition for a rule $L_\rho \leftarrow K_\rho \rightarrow R_\rho$ is an additional morphism (inclusion) $AC \leftarrow L_\rho$. If an application condition is a positive application condition, it is satisfied for a match $o : L_\rho \rightarrow AN$ if there exists also a valid match $AC \rightarrow AN$. This means that a positive application condition requires additional structures for a match to find. For example, for the three

[5] In the following, we simply use the transition name for describing the triggering action, e.g. "the host *quits*", instead of saying "the host client fires the transition *quit*".



(a) Skype AHLI net before application (b) After application of ExecuteCallRequest1

Figure 4: Example application of ExecuteCallRequest1 for *Alice* calling *Bob*

upper places in $L_{ExecuteCallRequest1}$ we can formulate an application condition consisting of the net in Fig. 1. This ensures that the match has to find e.g. a transition like *activate* in Fig. 1 between the places in *AN* that are matched by *User1* and *isParticipating1* with *o*. Similarly, we can forbid additional parts with negative application conditions, which demand that there does not exist a valid match $AC \rightarrow AN$.

We omit the actual positive application conditions for rules in this paper and express informally in the left-hand side with dashed boxes that their contents can be matched only on corresponding places of a client, e.g. as in the left-hand side of ExecuteCallRequest1 with the box around the three upper places and similarly with the box around the lower two places. Additionally, for a better understanding of the rule's effect, a dashed box for a client role in the rule's right-hand side comprises the transitions that are intended to represent new possible actions that can be performed by this particular client, e.g. the transition *quit* in the right-hand side of ExecuteCallRequest1 that is supposed to be a host action.

Inviting to a Conference The rule ExecuteCallRequest1 would be sufficient for modeling direct calls involving only two people, which we consider as a minimalistic conference. Now we extend the system by a rule that allows to invite more clients to a running conference. In this case, we employ the second rule ExecuteCallRequest2 in Fig. 5 for reacting to call requests and for inviting more clients. This rule needs to match a *Conferencing* place that is hosted by the requesting client (note the dashed box, representing an appropriate application condition) who has not already quit. On the one hand, this rule simply connects the requested participant (containing the lower boxed parts) to the existing conference as the previous rule would do if the conference were not already created. On the other hand, it deletes the transition *requestForward*, which restricts the behavior of the first callee that has been connected with ExecuteCallRequest1 before, because we allow call forwarding only in one-to-one calls.

The empty place *QuitRequest* in the left-hand side could perfectly match a place with a token *Control*, so we have to avoid a host to quit the conference before the system connects a newly invited participant by an explicit token value *False*.

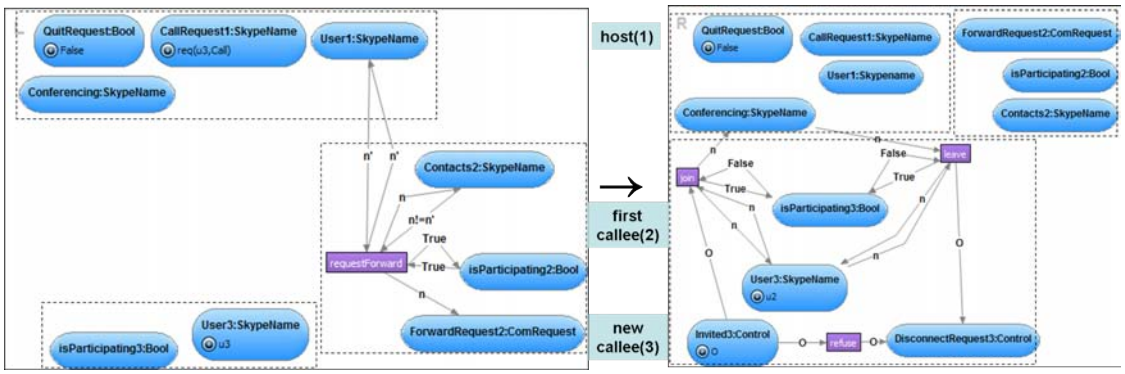


Figure 5: Rule ExecuteCallRequest2

Leaving a Conference The rule ExecuteDisconnectRequest in Fig. 6, which handles the request for disconnecting a called client, is almost the inverse of the rule ExecuteCallRequest2 in Fig. 5 but with a *Control* token on *DisconnectRequest* instead of *Invited*. It assumes that a client either has refused the connection or has joined and then left, which created a request token for disconnection. The rule then deletes the conference structure part of the client to be disconnected as they are no longer needed. When leaving a conference there is just no need to represent these actions any more. Note that the rule leaves the parts untouched that have been in the client from the start in order to leave the core client functional and to allow further connections.

For disconnecting the callee that has been connected first to a conference with ExecuteCallRequest1, a similar rule is needed that also deletes this participant's *requestForward* transition and *ForwardRequest* place.

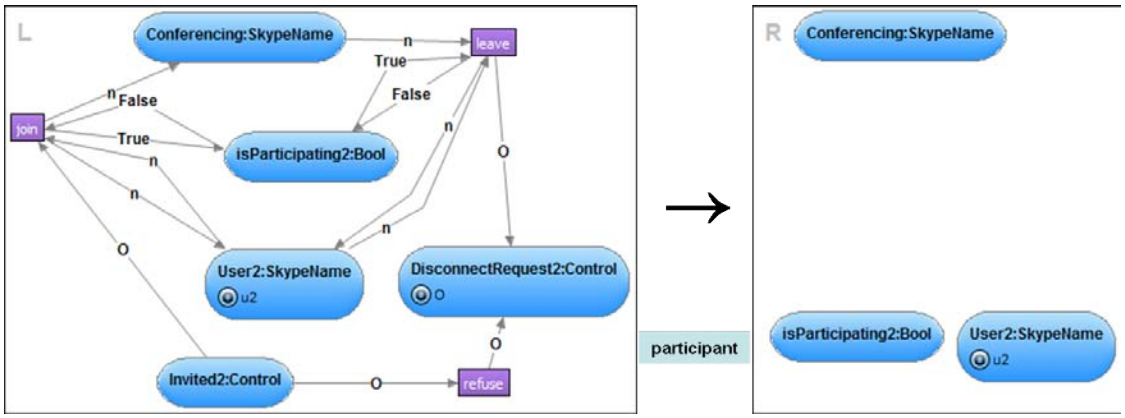


Figure 6: Rule ExecuteDisconnectRequest

Closing a Conference If the host has quit the conference, the rule ExecuteQuitRequest in Fig. 7 can match and remove the remaining conference structure after all participants have left the conference with the previous rule. We do not need a negative application condition to prevent "dangling" *join* and *leave* transitions from possibly connected clients. Due to the dangling conditions for rule applications, a rule can not delete the environment of a transition that is not part of the match. More precisely, a rule is not applicable if a matched place is connected to an unmatched transition and the rule deletes this place.

Further Use Cases for Reconfiguration Due to space limitations we only discuss briefly some more use cases for reconfiguring rules: A rule for executing the forward request of the callee in a one-to-one call deletes the transitions and places of the callee appearing in a conference and builds up the same structure for the client that the call is forwarded to so that it has the choice to join or refuse the call.

Another interesting aspect is the creation of clients. A rule with empty left hand side and the client net of Fig. 1 as right-hand side (but with a variable as *SkypeName* user token and empty *Contacts* place) could be used to create clients for new users. An application condition should

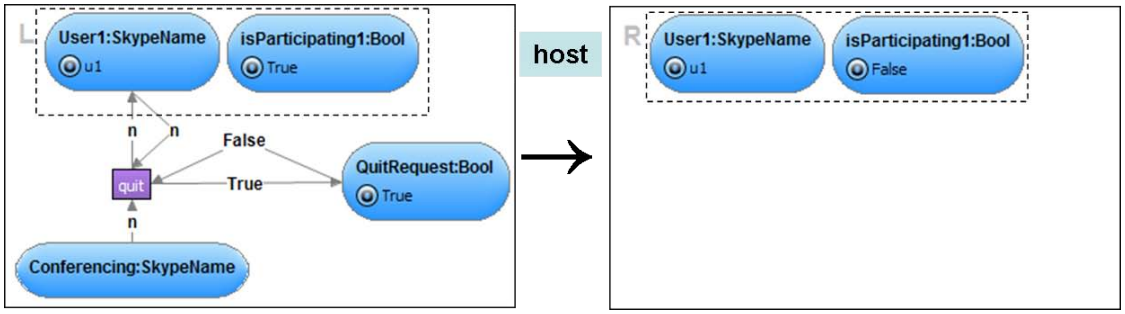


Figure 7: Rule ExecuteQuitRequest

prevent that a client with the same user name as an existing one is created by the rule.

For the previous rules, we assumed that the host accesses contacts on his *Contact* place. With an additional transition, the client could produce requests for contact exchanges that lets a client with the requested name decide whether the requester may add the unknown client's name to its contact list (cf. the rules given in [Mod10]).

Finally, one may object that there are cases when call requests should be discarded, e.g. when the called client is offline or does not want to be disturbed, which can be realized with appropriate negative application conditions for the executing rules and a rule that catches these illegal requests by deleting them.

2.3 Multicasting with Amalgamated Rules

In Skype conferences and chats, one client transmits data to a group of clients, i.e. the participating clients, which is a typical concept in communication platforms. This communication behavior is called multicasting and is a challenge for Petri net-based modeling techniques since the number of participating clients is not known a priori. A possible but not fully adequate way is to split the group transmission into single transmissions to each participating actor. This makes it necessary to code the status of the multicast data for each actor into the net. Furthermore, each client has to be connected to every other actor in order to be able to perform multicasting which leads to a massive explosion of the net structure size in communication nets with multicasting.

Instead, we use a more practical way of specifying multicasting for communication platforms [BEE+09], namely amalgamation of AHLI net transformation rules, which realizes a controlled, parallel rule application at all possible matches at once [Gol10]. Informally, the application of an amalgamated rule is the application of one or more rules (called multi-rules) that have a common subrule (called kernel rule) at several matches in parallel that overlap in a match of the kernel rule. An interaction scheme defines such a set of multi rules and relates them by a common subrule.

Consider a conference in the AHLI Skype net. Now, one of its participating clients, considered the sender in this situation, generates some data to be transmitted by firing its transition *send* (cf. Fig. 1). This produces a token of type *Data* on the *Out* place of this client component[6].

[6] Actually, *Data* may be any kind of data: textual, visual, acoustic etc.

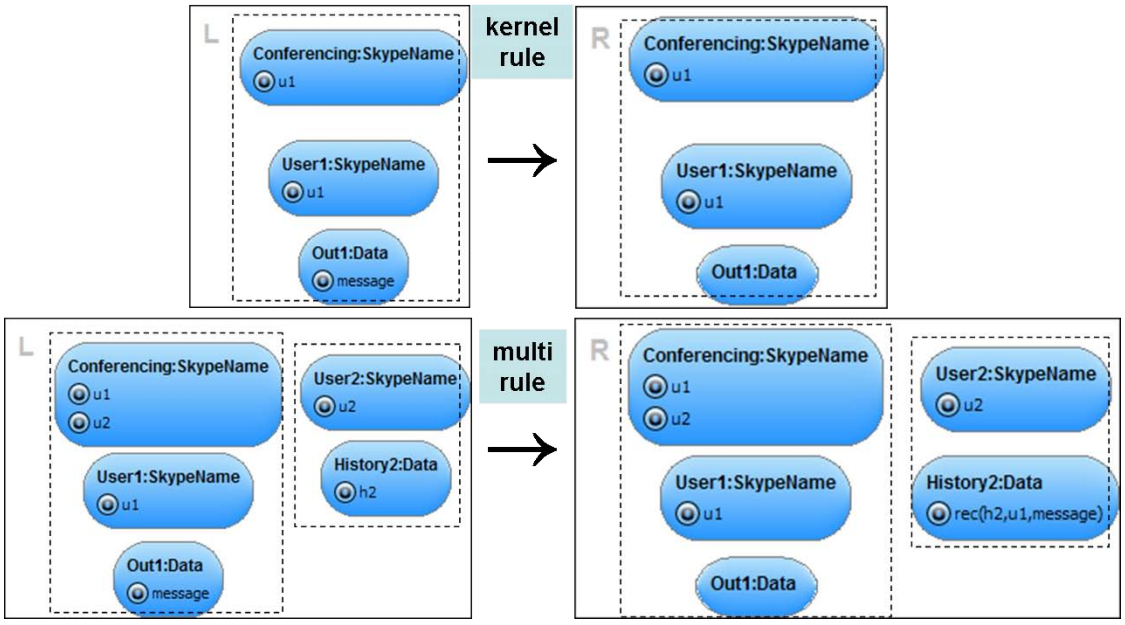


Figure 8: Interaction scheme for multicasting in conferences

Somehow, this data has to be dispatched to all other participating clients, which we realize with the interaction scheme consisting of the one multi rule and the kernel rule in Fig. 8.

The top part is the kernel rule that matches the sender client's name token on the conference place and its *Out* place with the data token to send. The only effect of the kernel rule is that the data token is deleted from the sender's client. A match for the kernel rule fixes the conference and the sender whose message is multicasted to all participating clients. A match for the multi rule now complies to the kernel match on the common parts and additionally matches another client that corresponds to another *SkypeName* token on the conference place and sends the message to the receiving client. Here we use an appropriate algebraic operation *rec* that combines the actual history *h2*, the sender *u1* and the message itself to generate the new history of messages.

We require that amalgamated rules over this interaction scheme are built with maximal matching, i.e. that for a given kernel match the multi rule is applied at all possible matches on the Skype AHLI net. Clearly, for every participant in a conference fixed by the kernel match there exists a match for the multi rule. Each match of the multi rule is part of the amalgamated rule and lets this participant receive the multicasted message when the amalgamated rule is applied. Thus, applications of amalgamated rules built over maximal matchings of the interaction scheme in Fig. 8 realize complete multicastings in conferences.

Further Use Cases for Amalgamated Rules Interaction schemes with maximal multi rule matching is an important and often used formalism to handle an unknown number of structural parts or tokens. For example, the deletion of a user client is not possible with a simple rule because there is an unknown number of name tokens on its *Contacts* place and a rule can only delete a place when all of its tokens are matched (cf. Subsect. 2.2). Another interesting case

would be an interaction scheme for executing more than one call request of a conference host with one transformation step, which, when amalgamated over all possible matches, connects all clients to the conference structure that correspond to the request tokens.

2.4 Chats

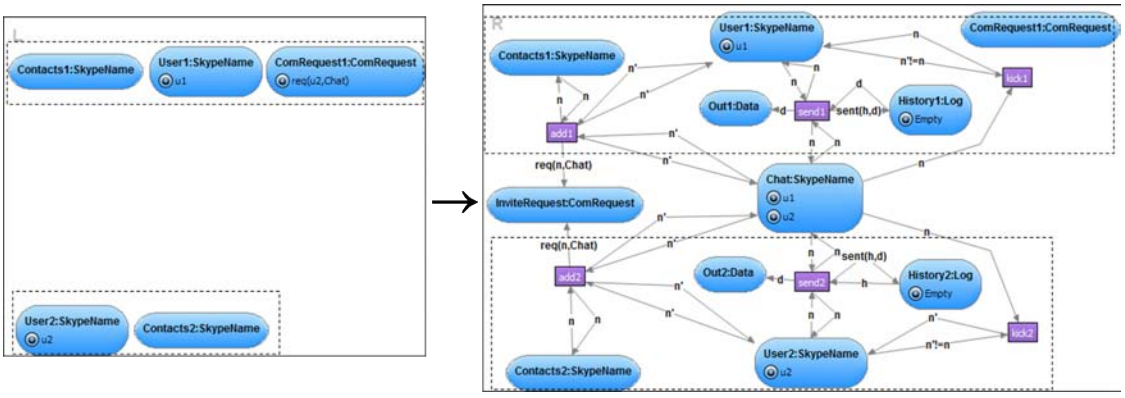
In Skype, a client can participate in many chats in parallel. This means that a client has to create the data to be transmitted and to log events in the context of the chat. The transition *send* in the client component in Fig. 1 is not adequate to create messages for chats because there is no way to specify the context of the message, i.e. the chat it is meant to be multicasted in. Moreover, the part of the history of a chat that a participant experienced is always available to him once the chat has started, even if he was kicked by another participant. We assume a further functional requirement for chats: If a client participated in a chat and has been kicked out, then its history should be preserved so that on reinvitation the client can continue its history for the same chat where it has left.

Creating Chats With the rule ExecuteCreateChat in Fig. 9(a), the system reacts to a request token $req(u2, Chat)$ on a client's *ComRequest* place by creating a chat structure similar to rule ExecuteCallRequest1 in Fig. 3 before. A place carries the names of all participating users and for the initiator and the invited user the rule creates transitions and places for sending and receiving data in the context of this chat.

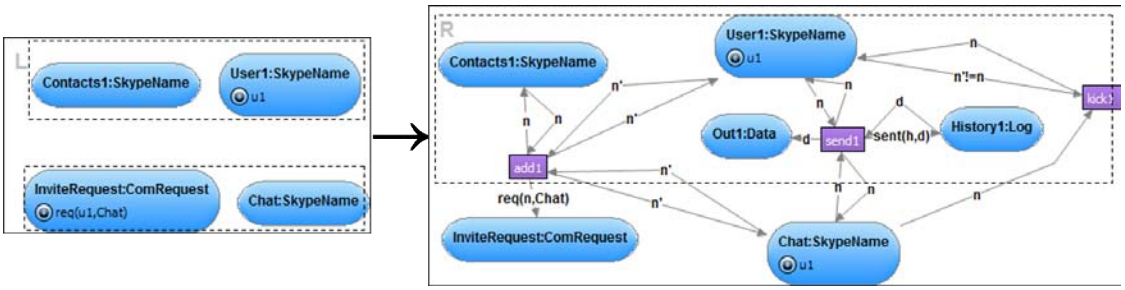
Multicasting in Chats For multicasting data tokens that a participating chat user has produced with the local *send* transition, we can employ an interaction scheme similar to the one for conferences in Fig. 8. The only differences are that the conferencing place is now a chat place and the application conditions have to ensure that the *Out* and *History* places associated to a participant are the local ones for this chat instead of the corresponding places in the client of Fig. 1.

Managing Chats In chats, every participant may kick other participants and add some of his contacts, as long as he has not already been kicked himself. In the two rules ExecuteCreateChat in Fig. 9(a) and ExecuteAddChat in Fig. 9(b) the corresponding transitions for these actions are created for clients that have been invited to a chat, where the rule ExecuteAddChat reacts to an invitation request that any chat participant has created.

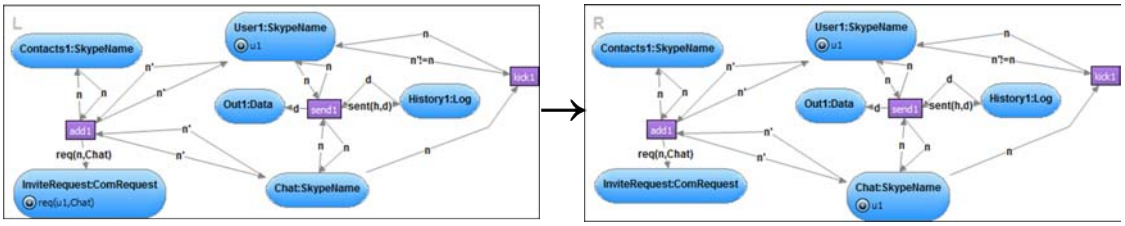
A kicked participant's name gets removed from the chat place but the structure around his local chat transition *send* will be preserved and so will his history. Nevertheless, henceforth this user can not send or receive messages in this chat. If a kicked user of a chat gets readded, the rule ExecuteReaddChat in Fig. 9(c) can be applied, which assumes that the local chat structure for a client is already present and simply readds its user name to the chat place. As a result, the history of a kicked participant is preserved and the client again can send messages and receives messages from applications of the multicasting interaction scheme. Finally, in order to avoid redundant structure to be created, we have to prevent the rule ExecuteAddChat to be applied in the case when a client is already connected to a chat. We realize this by defining the left-hand side of rule ExecuteReaddChat as a negative application condition for rule ExecuteAddChat.



(a) Rule ExecuteCreateChat



(b) Rule ExecuteAddChat



(c) Rule ExecuteReaddChat

Figure 9: Rules for creating and managing chats

3 Analysis of Reconfigurable AHLI Net Models

In this section, we first state that AHL nets with individual tokens are an $\mathscr{M}$-adhesive category and summarize the main results available for reconfigurable AHLI nets. The proof of these technicalities can be found in [MGE+10]. Afterwards we show that a firing step can be simulated by an appropriate rule and present an important result concerning the independence of firing steps and rule applications.

3.1 Petri nets with Individual Tokens compared to Collective Tokens

The main result in [EEPT06] concerns the formal foundation for transformations of low-level and high-level Petri nets in the collective token approach based on the framework of $\mathscr{M}$-adhesive categories, i.e. weak adhesive high-level replacement (HLR) systems in the sense of [EHPP06]. $\mathscr{M}$-adhesive categories have been introduced as a new categorical framework for graph transformation in the DPO approach. They combine the well known framework of HLR systems with the framework of adhesive categories introduced in [LS05].

The framework of $\mathscr{M}$-adhesive categories provides many useful results well-known from graph transformation [Roz97], concerning the applicability of rules, embedding and extension of transformations, parallel and sequential dependence and independence, and concurrency of rule applications. The concept of parallel independence states that two parallel transformation steps are not in conflict, i.e. each of the transformations does not delete anything that is required by the other one. Two consecutive transformation steps are sequentially independent if they are not causally dependent, i.e. the first rule does not produce anything that is required by the second. Provided that the corresponding conditions are satisfied two alternative transformation steps can be swapped and each of them is applicable after the other one.

Classical Petri nets and AHL nets have been shown in [EEPT06, EHP+08] to define a weak adhesive HLR category for the class $\mathscr{M}$ of all injective net morphisms. This allows us to apply all the results for $\mathscr{M}$-adhesive systems shown in [EEPT06] also for Petri net transformation systems. The concept of Petri systems leads to a category **PTSys** with morphisms allowing to increase the number of tokens on corresponding places. Unfortunately, $(\mathbf{PTSys}, \mathscr{M}_{inj})$ with the class $\mathscr{M}_{inj}$ of all injective morphisms is not $\mathscr{M}$-adhesive in contrast to $(\mathbf{PTSys}, \mathscr{M}_{strict})$, where $\mathscr{M}_{strict}$ is the class of strict injective morphisms where the number of tokens on corresponding places is equal. This, however, is an unpleasant restriction for the usability of the transformation approach, especially the firing of a transition cannot be simulated in a natural way by the application of a corresponding "transition rule". Since we have shown that AHLI nets form an $\mathscr{M}$-adhesive category with a class of non-strict morphisms $\mathscr{M}$, we can apply these results to the rules presented in Sect. 2. Amalgamated rules are in general standard rules in $\mathscr{M}$-adhesive transformation systems so that we have the same results for them [Gol10].

An example for parallel independent transformation steps is given by the rules *ExecuteCallRequest2* (cf. Fig. 5) and *ExecuteDisconnectRequest* (cf. Fig. 6) applied to the same clients, where Alice is hosting a conference in which both Bob and Carol are currently participating and there is a call request for Dave. In this case we can first apply *ExecuteDisconnectRequest* to disconnect Carol's client, as well as Dave can be invited first by applying *ExecuteCallRequest2*. These requests are parallel independent because each of these rules can be applied after the application of the other yielding the same result where Carol's client has been disconnected and Dave has been invited.

3.2 Simulating Firing Steps by Rule Applications

Based on the observation of parallel and sequential independence of rule applications the main results in [EHP+07] deals with conflict situations between transformations and token firing. The traditional concurrency situation in Petri nets is that two transitions with overlapping pre domain

are both enabled and together require more tokens than are available in the current marking. As AHLI nets can evolve in two different ways the notions of conflict and concurrency become more complex. Assume that a given AHLI net represents a certain system state. The next evolution step can be obtained not only by token firing but also by the application of one of the rules available. Hence under certain conditions each of these evolution steps can be postponed after the realization of the other, yielding the same result, and can be performed in a different order without changing the result.

In the individual token approach we are able to manipulate the net's marking by rule application. Thus, here the result of independent firing and transformation steps is achieved in a more elegant way by using the well-known result of two independent transformation steps. For this reason we first construct a specific rule which application corresponds to a single firing step. The so-called AHLI transition rule consists of a fixed AHLI net part given by a transition and its environment. The marking of the AHLI net in the left hand side is chosen in such a way that the transition is enabled and the marking of the AHLI net in the right hand side is exactly the follower marking after transition firing, while the marking in the interface is empty. Note, that the construction of an AHLI transition rule only depends on a specific transition with corresponding token selection. But there may exist several consistent transition assignments enabling the transition and, therefore, different consistent transition assignments may result in the same AHLI transition rule. In [MGE+10], we have shown that there is an equivalence between a firing step of an AHLI net and a transformation step via the corresponding AHLI transition rule.

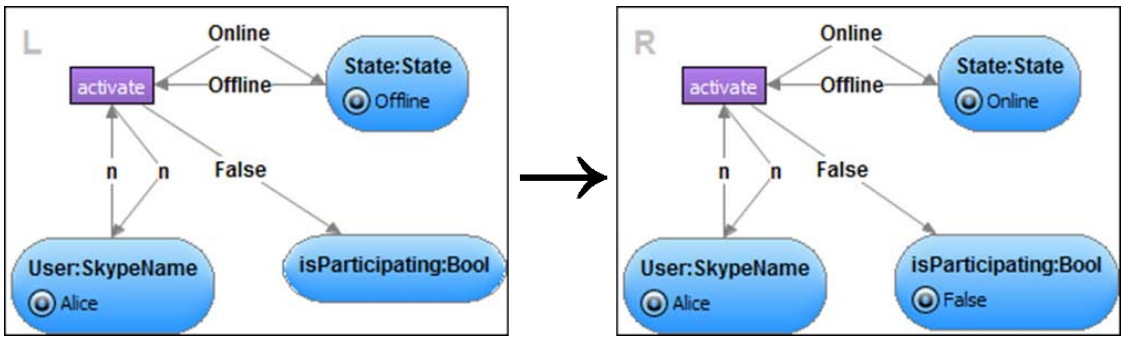


Figure 10: Example Transition Rule

An AHLI transition rule is depicted in Fig. 10: The fixed AHLI net part for L, K, and R is given by the transition *activate* and its environment consisting of the places *State*, *User* and *isParticipating* as well as the net inscriptions according in Fig. 1. As explained in Subsect. 2.1 the tokens with values *Offline* and *Alice* in the left-hand side L enable the transition *activate* and are replaced by tokens with values *Online*, *Alice*, and *False* on the corresponding places in the right-hand side R while the marking of the interface K is empty.

4 Higher-Order Nets as Control Structures

In this section, we discuss an extension of AHLI nets to support an adequate control structure over complex Skype systems and their rule applications.

Due to our case study modeling Skype with AHLI nets and AHLI transformation rules, we have proven that this technique is powerful enough to model typical features of communication platforms. Anyway, the concepts of Petri net behaviour via transition firing and Petri net reconfiguration with transformation rules are only loosely coupled. We designed the rules to demand requests to be applicable to the Skype AHLI net for realizing the requests with reconfiguration of the Skype net, but the formalism of reconfigurable AHLI nets itself is more similar to graph transformation systems in the sense that rules are assumed to be applied "at the right time". So, we need a possibility to express that requests must be processed immediately, i.e. a control structure for the firing behavior and reconfiguration of AHLI nets. Several formalisms for controlling graph transformations have been proposed [HEET98, KK99] but due to their orientation to graph transformation they lack the possibility to relate reconfigurations (by rule applications) to firing steps.

Algebraic Higher-Order Nets A useful approach for controlling firing steps and transformations of low-level Petri nets has been introduced in [HME05] with AHL nets that contain place/transition nets and transformation rules as tokens. In our project forMA_lNET, we call such nets algebraic higher-order (AHO) nets due to the "nets on nets" structure. AHO nets have two important aspects that distinguish them from the original notion of nested Petri nets called object nets in [Val98]: First, AHO nets do not only have Petri nets as tokens but also transformation rules, which qualifies them especially for controlling reconfigurable Petri nets. Second, AHOs are formulated as regular AHL nets with signatures and algebras that provide operations for firing the token nets and for applying token rules on token nets. This means, that we do not need a new formalism and can use all results for AHO nets that we already have for AHL nets. Moreover, we can use any kind of Petri nets and their corresponding transformation rules as tokens, as long as we can define their firing behavior and transformation algebraically for the containing AHO net.

We recapitulate the example from [HME05] that realizes reconfigurable Petri nets as an algebraic higher-order net and then discuss an idea how we can use this approach for better controlling request execution in our case study.

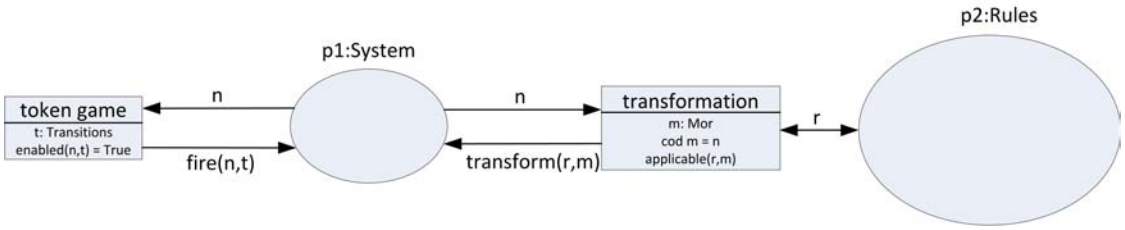


Figure 11: Algebraic Higher-Order Net for Reconfigurable Petri Nets

Fig. 11 depicts an AHO net where we assume that it can carry marked place/transition nets

as tokens of the sort *System* on its place p_1 and Petri net transformation rules as tokens of the sort *Rules* on its place p_2. To simulate a firing step of a net token on p_1, the AHO net fires its transition *token_game*, which selects a net token n from p_1 if this net contains some enabled transition, which is ensured by the firing condition $enabled(n,t) = True$ of the higher-order transition. The firing of *token_game* then produces a token with value $fire(n,t)$ that is just the net n where t has fired.

Rule applications are realized similarly by firing the higher-order transition *transformation*, which selects a net token n from p_1 and a rule token r from p_2 if the rule is applicable at some match m into net n. Firing *transformation* then produces the token net that results from the application of the selected rule on this match, which is calculated algebraically by $transform(r,m)$.

Controlling Skype AHLI nets We now consider two extensions of the approach in [HME05] in order to express with an AHO net that actions in Skype AHLI nets posing requests must be handled immediately by reconfiguring rule applications. Obviously, we have to extend the definitions of the sorts *System* and *Rules* (cf. Fig. 11) from place/transition nets and rules to AHLI nets, AHLI rules and interaction schemes with maximal matching, respectively. Then we can build an AHO net for the Skype case study as in Fig. 12. The transition *token_game* now only fires transitions in the token net *Skype* that are not producing a request token or can be understood as a signaling request in any way. This is decided by the predicate $produces\_request(n,t,asg)$ that has to be defined according to the system to be modeled. Note that for high-level nets as tokens we also need to consider a valid variable assignment for a firing step. The other transitions in the token net that may trigger a reconfiguring rule, i.e. with $produces\_request(n,t,asg)$ evaluating to *true*, are fired in the token net *Skype* via the higher-order transition *request_and_execute*, which first fires a transition in the token net n and then immediately applies a rule from p_2 on n. With this AHO net, we can express the complete firing and reconfiguration behavior of the case study in a single AHL net and requests in the Skype systems can not remain unexecuted.

This is just a first step of generalizing reconfigurable nets. We are free to distribute the transformation rules to different places and to restrict their applications by conditions that are expressed by algebraic operations depending on the system we want to model, like we did with *produces_request* in the small example. With AHO nets we can ensure that these conditions are met and for example relate transition firing steps with rule applications, which is a further valuable tool for modeling and proving properties in addition to the analysis results in the previous section.

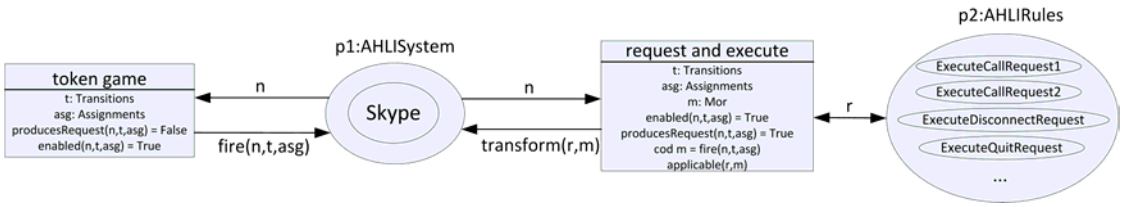


Figure 12: Algebraic Higher-Order Net for Skype AHLI Nets

5 Conclusion

In this paper, we have shown that our approach of algebraic high-level nets with individual tokens is powerful enough to cover the main requirements of communication platforms by examplarily modeling the network's topology and the group communication in Skype. In our approach, structures for communication are added to system only in case when they are needed and are removed after the communication action has been completed and we strictly distinguish user triggered client behavior and system reactions. We have presented the important result of the category of AHLI nets forming a $\mathscr{M}$-adhesive category to analyze AHLI models for communication platforms e.g. by parallel and sequential independence.

An essential aspect that is used frequently in communication platforms is multicasting, i.e. transmission of data to selected actors, which can not be realized adequately only with the classical firing behavior of Petri nets [BEE+09]. For modeling multicasting, we refrain from adding a massive net structure that connects the sender with each of the receiver components. Instead, we defined an interactions scheme allowing to find all receivers with the appropriate access rights in a possibly large system net. For our approach of modeling multicasting with amalgamated rule applications [BFH87], we rely on the formalism of nets with individual tokens, which allows to formulate rules for $\mathscr{M}$-adhesive systems with that can manipualte freely the marking of nets, in contrast to $\mathscr{M}$-adhesive systems of nets with "collective" markings.

In order to control the firing and reconfiguration of AHLI models for communication platforms we proposed to use the AHLI formalism itself by considering the actual net and the rules of a reconfigurable AHLI system to be tokens in an algebraic higher-order net. The means, for controling the firing and reconfiguration behavior of their net and rule tokens, suitable operations are defined in the higher-order net's algebra for firing transitions in the net tokens and for applying the rule tokens on the net tokens [HME05]. It remains to examine how the structure of the higher-order net can be further used to improve the modeling of communication platforms. Another possible abstraction is the step to reconfigurable AHO nets, which could be used for specifying system updates and changes by creating new rule tokens or modifying existing rules.

We have implemented an Eclipse-based tool environment, which currently allows modeling, simulation and analysis of a restricted kind of AHO nets with P/T nets and rules as tokens [BM08]. This tool supports analysis of all possible types of conflicts w.r.t to a token net: between applications of two rules on a net token, between firing steps in a net token, and between firign steps and applications of a rule on a net token (cf. Subsect. 3.2). An extension of our tool to reconfigurable AHLI nets with amalgamation and controlling AHO nets that supports the presented approach is planned for the future.

Bibliography

[Agh85] G. Agha. *ACTORS: A Model of Concurrent Computation in Distributed Systems*. PhD thesis, MIT, 1985. Cambridge: MIT Press.

[Bau90] B. Baumgarten. *Petrinetze, Grundlagen und Anwendungen*. BI, 1990.

[BEE+09] E. Biermann, H. Ehrig, C. Ermel, K. Hoffmann, T. Modica. Modeling Multicasting in Dynamic Communication-based Systems by Reconfigurable High-level Petri Nets. In *IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2009, Corvallis, OR, USA, 20-24 September 2009, Proceedings*. Pp. 47–50. IEEE, 2009.

[BFH87] P. Böhm, H.-R. Fonio, A. Habel. Amalgamation of graph transformations: a synchronization mechanism. *Journal of Computer and System Science*, pp. 377–408, 1987.

[BM08] E. Biermann, T. Modica. Independence Analysis of Firing and Rule-based Net Transformations in Reconfigurable Object Nets. In C. Ermel and Heckel (eds.), *Proc. Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT'08)*. Volume 10. EC-EASST, 2008.

[EEPT06] H. Ehrig, K. Ehrig, U. Prange, G. Taentzer. *Fundamentals of Algebraic Graph Transformation*. EATCS Monographs in Theoretical Computer Science. Springer Verlag, 2006.

[EHP+02] H. Ehrig, K. Hoffmann, J. Padberg, P. Baldan, R. Heckel. High-Level Net Processes. In *Formal and Natural Computing*. LNCS 2300, pp. 191 – 219. Springer, 2002.

[EHP+07] H. Ehrig, K. Hoffmann, J. Padberg, U. Prange, C. Ermel. Independence of Net Transformations and Token Firing in Reconfigurable Place/Transition Systems. In *Proc. of 28th International Conference on Application and Theory of Petri Nets and Other Models of Concurrency*. LNCS 4546, pp. 104–123. Springer, 2007.

[EHP+08] H. Ehrig, K. Hoffmann, J. Padberg, C. Ermel, U. Prange, E. Biermann, T. Modica. Petri Net Transformations. In *Petri Net Theory and Applications*. Pp. 1–16. I-Tech Education and Publication, 2008.

[EHPP06] H. Ehrig, A. Habel, J. Padberg, U. Prange. Adhesive High-Level Replacement Systems: A New Categorical Framework for Graph Transformation. *Fundamenta Informaticae* 74(1):1–29, 2006.

[EM85] H. Ehrig, B. Mahr. *Fundamentals of Algebraic Specification 1: Equations and Initial Semantics*. EATCS Monographs on Theoretical Computer Science 6. Springer, 1985.

[EPR93] H. Ehrig, J. Padberg, L. Ribeiro. Algebraic high-level nets: Petri nets revisited. In *Proc. of the ADT-COMPASS Workshop'92 (Caldes de Malavella, Spain)*. Berlin, 1993. Technical Report 93-06.

[Gol10] U. Golas. Multi-Amalgamation in $\mathscr{M}$-Adhesive Categories. Technical report 2010/05, Technische Universität Berlin, 2010.

[HEET98] R. Heckel, H. Ehrig, G. Engels, G. Taentzer. Classification and comparison of modularity concepts for graph transformation systems. In *Proc. 6th Int. Workshop on Theory and Application of Graph Transformation (TAGT'98)*. 1998.

[HME05] K. Hoffmann, T. Mossakowski, H. Ehrig. High-Level Nets with Nets and Rules as Tokens. In *Proc. of 26th Intern. Conf. on Application and Theory of Petri Nets and other Models of Concurrency*. LNCS 3536, pp. 268–288. Springer, 2005.

[Jen92] K. Jensen. *Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use*. Volume 1: Basic Concepts. Springer Verlag, EATCS Monographs in Theoretical Computer Science edition, 1992.

[Jen94] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*. Volume 2: Analysis Methods. Springer, EATCS Monographs in Theoretical Computer Science edition, 1994.

[Jen97] K. Jensen. *Coloured Petri Nets - Basic Concepts, Analysis Methods and Practical Use*. Volume 3: Practical Use. Springer, EATCS Monographs in Theoretical Computer Science edition, 1997.

[JR91] K. Jensen, G. Rozenberg (eds.). *High-Level Petri Nets*. Springer, 1991.

[KK99] H.-J. Kreowski, S. Kuske. Graph Transformation Units with Interleaving Semantics. *Formal Aspects of Computing* 11:690–723, 1999.

[LS05] S. Lack, P. Sobocinski. Adhesive and Quasiadhesive Categories. *Theoretical Informatics and Applications* 39(5):511–546, 2005.

[MEE+10] T. Modica, C. Ermel, H. Ehrig, K. Hoffmann, E. Biermann. Modeling Communication Spaces with Higher-Order Petri Nets. In Lasker and Pfalzgraf (eds.), *Advances in Multiagent Systems, Robotics and Cybernetics: Theory and Practice*. Volume III. The International Institute for Advanced Studies in Systems Research and Cybernetics, Tecumseh, Canada, 2010. to appear.

[MGE+10] T. Modica, K. Gabriel, H. Ehrig, K. Hoffmann, S. Shareef, C. Ermel, F. Hermann, U. Golas, E. Biermann. Low and High-Level Petri Nets with Individual Tokens. Technical report 13/2009, Technische Universität Berlin, 2010.

[Mil99] R. Milner. *Communicating and Mobile Systems: the Pi-Calculus*. Cambridge University Press, June 1999.

[MM89] N. Martí-Oliet, J. Meseguer. From Petri Nets to Linear Logic. In LNCS389 (ed.), *Proc. Category Theory and Computer Science*. Pp. 313–340. Springer-Verlag, 1989.

[MM90] J. Meseguer, U. Montanari. Petri Nets are Monoids. *Information and Computation* 88(2):105–155, 1990.

[Mod10] T. Modica. Towards Formal Algebraic Modeling and Analysis of Communication Spaces. In Haveraaen et al. (eds.), *CALCO Young Researchers Workshop (CALCO-jnr 2009) – Selected Papers*. Technical Report 5-2010, pp. 89–103. Università di Udine – Dipartimento di Matematica e Informatica, 2010.
http://calco09.dimi.uniud.it/calcojnr_booklet.pdf

[PER95] J. Padberg, H. Ehrig, L. Ribeiro. Algebraic High-Level Net Transformation Systems. *Mathematical Structures in Computer Science* 5:217–256, 1995.

[Pet62] C. Petri. *Kommunikation mit Automaten*. PhD thesis, Schriften des Institutes für Instrumentelle Mathematik, Bonn, 1962.

[Rei85] W. Reisig. *Petri Nets: An Introduction*. EATCS Monographs on Theoretical Computer Science 4. Springer, 1985.

[Roz97] G. Rozenberg. *Handbook of Graph Grammars and Computing by Graph Transformations, Volume 1: Foundations*. World Scientific, 1997.

[RT86] G. Rozenberg, P. Thiagarajan. Petri Nets: Basic notions, structure, behaviour. In *Current Trends in Concurrency*. LNCS 224, pp. 585–668. Springer, 1986.

[Val98] R. Valk. Petri Nets as Token Objects: An Introduction to Elementary Object Nets. In *ICATPN '98: Proceedings of the 19th International Conference on Application and Theory of Petri Nets*. LNCS 2987, pp. 1–25. Springer, 1998.

[Win87] G. Winskel. Petri nets, algebras, morphisms, and compositionality. *Information and Computation* 72:197–238, 1987.