Towards Formal Algebraic Modeling and Analysis of Communication Spaces*

Tony Modica

modica@cs.tu-berlin.de
Integrated Graduate Program Human-Centric Communication
Technische Universität Berlin

Abstract. We subsume Communication Spaces (CS) as communication-based systems taking into account the central notions of interpretation of content in contexts, communication roles, and allowing for human-centric demands, e.g. adaption to environment and preferences. Since most of the well-known formal modeling approaches are adequate only for specific aspects or limited views of systems considered as CS, in this article a new formal approach is advocated. This approach is an integration and extension of the well-established modeling techniques of algebraic high-level Petri nets and rule-based graph transformation, intended to cover the main aspects of CS and to analyze and verify properties specific to them. We demonstrate the new approach of Algebraic Higher-Order Net with Individual Tokens (AHOI nets) on an example modeling of the widely known Internet telephone software Skype. This allows us to discuss the advantages of AHOI nets w.r.t. needs of a basic modeling of CS.

1 The Challenges of Communication Spaces

The notion of *Communication Spaces* (CS) is not intended to be fixed formally. Moreover, it is meant to serve as a characterizing concept of communication systems featuring specific aspects, e.g.:

- *Contextuality* of contents that is transmitted (via *channels*) by communicating entities (*actors*).
- *Dynamics* in the system structure, so that actors may move in CS, even join or leave several different CS.
- *Preferences*, *access rights*, and *roles* are to be respected.

Typical examples that can be considered from the CS viewpoint are Internet-based applications like Skype, Facebook, or SecondLife; and also Mobile Ad-hoc networks or SmartHomes, in which appliances are connected intelligently

* This work has been supported by the Integrated Graduate Program on Human-Centric Communication at TU Berlin (http://www.h-c3.org/ra_en.html#RAE) and by the research project forMAlNET (http://tfs.cs.tu-berlin.de/formalnet) of the German Research Council

to offer increased comfort to their inhabitants. It is desirable to have a formal modeling technique for CS, so that we can specify the features of such systems unambiguously and are able to simulate, test, and analyze/verify[1] them, using the formal semantics of the modeling technique. We have observed that most of the well-known modeling techniques like UML and actor systems [1] or formal specification techniques like process algebras [2], low-level and high-level Petri nets [3,4], algebraic specification [5] and graph transformation [6], and different kinds of logic are only adequate to model and/or analyze specific aspects of CS. Plain Petri nets for example have a static structure. Graph transformation systems in contrast are dynamic in their structure but lack a description of system behaviour. Of course, appropriate graph transformation systems may also be used to simulate e.g. the behaviour of Petri nets but it seems advisable to distinguish system behaviour from reconfiguration and to possibly use standard results for analysis. Another thing to mention is that we believe that visual diagrammatic models as Petri nets and graphs can have advantages for system modeling w.r.t readability and understandability, though there is no standard measure for these properties.

As a consequence of the above, we advocate a new integrated formal modeling technique for CS. In this paper, however, we only give an informal introduction to our new visual modeling technique and demonstrate how it can be used to model a typical example of a CS: the Internet telephone software Skype.

2 Modeling of CS with a new Kind of Petri Nets

In this section, we formulate some main requirements for modeling of CS and give an informal overview to a modeling approach, called *AHOI Petri nets*.

2.1 Requirements for Modeling of CS

An adequate formal modeling approach would have to cover at least three main aspects of CS:

1. Data and content in the CS and the knowledge of actors (their *content spaces*).
2. Structure of the CS, which actors are connected to each other (in short, the *topology*). In particular, the structure should be dynamic to allow actors to enter and leave.
3. *Interactions* in and between different CS, transmission of data

According to our experience, no single classic modeling approach is powerful enough to achieve this. For this reason, we propose a new integrating approach: *reconfigurable **A**lgebraic **H**igher-**O**rder Petri nets with **I**ndividual tokens* (AHOI nets).

[1] General interesting properties are realted to consistency, safety and security requirements, liveness, termination etc.

Definition 1 (Algebraic Higher-Order Nets with Individual Tokens). *An Algebraic Higher-Order Net with Individual Tokens (AHOI net) is a tuple* (AN, I, m) *where*

- *AN is an Algebraic High-Level (AHL) Net as in [7] whose algebra features a suitable token sort for some kind of Petri net and operations for calculating activated transitions and the results of firing steps. The eponymous example with low-level place/transition nets as tokens are the Algebraic Higher-Order nets introduced in [8][2].*
- I *is the (possibly infinite) set of individual tokens of the AHOI net.*
- $m : I \to (A_{AN} \times P_{AN})$ *is the marking function, assigning each individual token to a pair* (a, p)*, representing an algebraic value* a *on place* p *in* AN*.*

A reconfigurable AHOI net is an AHOI net with a set of transformation rules. The main advantage of Petri nets with individual tokens to Petri systems following the collective token approach (where marking are elements of the commutative monoid $(A_{AN} \times P_{AN})^{\oplus}$) is that we can formulate transformation rules in the sense of the double-pushout (DPO) approach in [6], so that these rules also can change markings. In regular AHL nets, i.e. with collective token markings, changing of markings via DPO transformation is not possible due to technical restrictions. The firing behaviour of an AHOI net (AN, I, m) with finite I is equivalent to the behaviour of AN with the marking $\sum_{i \in I} m(i)$.

As we will see in the next section's example, AHOI nets are powerful enough to cover the main aspects of CS by integrating Petri nets (topology), abstract data types (content spaces), and net transformation (interaction and dynamics). However, to be able to follow this example, we first give an informal introduction to AHOI nets in a simplified notation, while a formal theory of AHL nets with individual tokens is under development in a technical report to appear.

2.2 AHOI Nets in a Nutshell

Instead of reviewing Algebraic High-Level nets we will go through a quick tutorial of AHOI nets, for which we avoid the formal notation and consider a simplified visual representation: Fig. 1 shows an AHOI net component, with rectangles as *transitions* and ovals as *typed places* that contain *tokens*. Arcs inscripted with terms connect places and transitions to form a bipartite graph. For example, there are the places *User* of type *SkypeName*, *Template* of type *DataUnit*, and *State* of type *State*, containing e.g. the tokens *Alice* and *Offline*, which represent values of the correspondent data types. Especially, we have a token *DU(Alice)* being itself a Petri net[3], which we consider as *higher-order* tokens (or *object nets*, to distinguish them from the containing *system net*). These three places constitute the *predomain* of transition *activate* because of the arcs pointing to

[2] The idea of nets in nets stems from [9] as an approach to represent mobile agents in a Petri net

[3] This net is represented as a value of an appropriate algebraic type for Petri nets.

the transitions from these places. Similarly, *User*, *Online*, *Ready2Talk*, and *Template* are the *postdomain* of *activate*, because of the arcs pointing to them from *activate*.

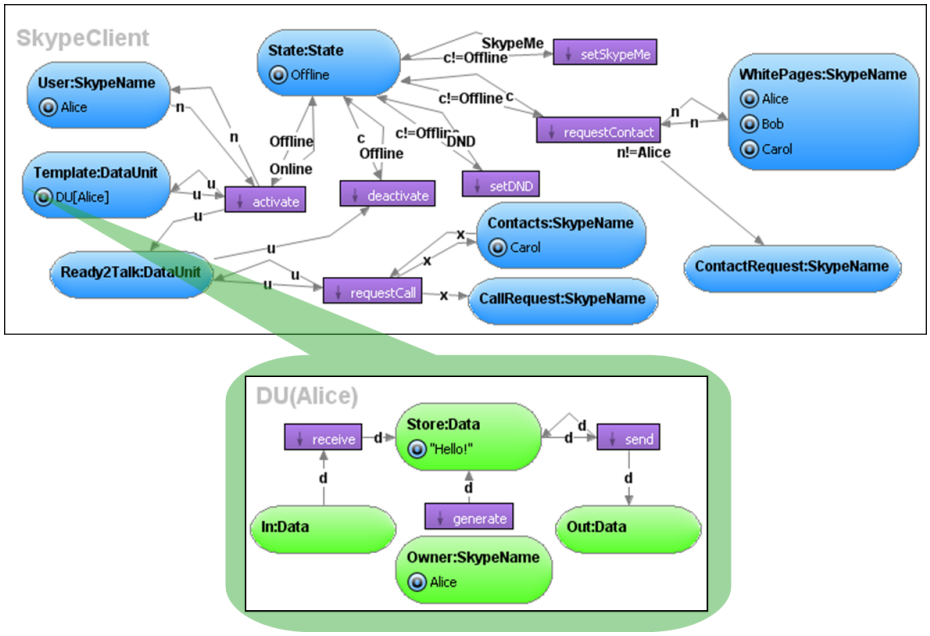

Fig. 1. AHOI net component for a Skype client

We call a transition *enabled* if on each of its predomain places we can find a token so that the set of selected tokens is consistent with the arc inscriptions. Basically, a token selection is consistent if the tokens correspond with the definite arc terms and we can find a consistent assignment for the arc variables to the selected tokens[4]. In Fig. 1 for example, activate is enabled because we can assign *u=DU(Alice)*, *n=Alice*, and the token *Offline* corresponds directly to the arc's term. If a transition is enabled, we can *fire* it, which means that the predomain tokens are removed and for each term on the postdomain arcs, a token according to the variable assignment and arc inscriptions is added to the corresponding place.

[4] Actually, in general AHOI nets arcs can have arbitrary terms with variables, e.g. $op(y, z, \ldots)$, and transitions additionally may have firing conditions like $op_1(x) = op_2(y, z, \ldots)$ according to the signature of the system net's algebra. But for the example, we only consider definite values and variables as arc inscriptions, as well as the simple condition pattern *v!=Val*, denoting that additionally the token value *Val* can not be assigned to variable v.

Note that e.g. place *User* is in the predomain and also in the postdomain of *activate* with the same arc inscription on the in- and outgoing arc, so its marking will not change on firing *activate*. Due to the arc to *Ready2Talk* inscripted with the same variable u as the arcs from and back to *Template*, firing *activate* copies the token assigned to u to both places. In short, if we fire *activate*, the result will be that *DU(Alice)* will have been copied to *Ready2Talk* and the token *Offline* on *State* will be replaced by token *Online*.

To reconfigure AHOI nets we use the rule-based net transformation approach in [6,10]. Transformation rules are spans of morphisms $L \leftarrow K \rightarrow R$ with a left-hand side L, a right-hand side R, and an interface K being the intersection of L and R, all these being AHOI nets. In Fig. 2, we consider an example AHOI net rule $p : L \leftarrow K \rightarrow R$ to demonstrate transformation: In general, to apply p to a net N we need to find an occurrence of L in N, specified by an injective AHOI net morphism $o : L \rightarrow N$. To get the context C, we remove everything that is matched by parts of L and is not preserved in K and finally add in the result net N' everything found in R that is not already present in the interface K. Altogether, Fig. 2 shows a net transformation via rule $p : L \leftarrow K \rightarrow R$, where N can be considered as gluing of L and C along K and N' as gluing of R and C along K[5]. Note that also the markings of places *p1* and *p3* are changed by this rule application.

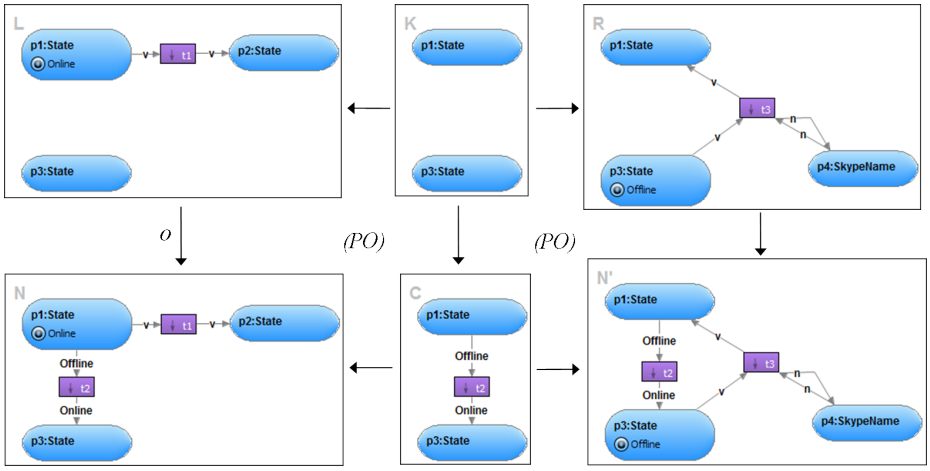

Fig. 2. Example AHOI net transformation

AHOI net morphisms (e.g. the match) must fulfill some structural conditions to ensure that rule applications yield valid AHOI nets [11]. Because L and R

[5] The resulting squares in Fig. 2 are pushouts in the category of AHOI nets, i.e. N is the gluing of L and C, where the interface K specifies items to be identified; analogously for N'.

are usually sufficient to understand which parts a rule deletes and creates, we mostly omit K and denote a rule as $L \to R$ in a compact notation.

3 Modeling Skype with AHOI Nets

Skype[6] is a widely used program for Internet telephony, offering easy to use (synchronized) data exchange and conferences. With its contact and privacy management, users can decide who and how other users can contact them. We discuss how Skype, as a typical representative offering many CS-relevant features, can be modeled with AHOI nets. First, we have to make some general decisions about what aspects we focus on and how to represent these.

Skype is not open source, there is no (publicly available) formal model, especially no one according to the CS viewpoint, and Skype uses proprietary network protocols. Therefore, we limit ourselves to modeling observable behavior only, i.e. to activities users can perform in their Skype client software and the direct effects of these activities caused in the Skype system. Our example follows these guidelines:

- A single AHOI net models the whole Skype system. Each user, resp. his client instance, is represented by an (initally discrete) component of this system net.
- We strictly distinguish user-triggered client behavior and system reactions. User actions are modeled by transitions in the corresponding client component; a user can act if at least one of its client's transitions is enabled. In contrast, global system actions are modeled by rules that reconfigure the system. To be more specific: An action in a client can either alter the client's configuration directly (like (de)activating the client, modifying privacy settings etc.) or represent a request to the Skype system to perform a global task (like establishing connections or transmitting data) that may extend/restrict possible actions of the client. System operations executing such requests are realized by rule applications, which possibly create or remove transitions of a client net component.
- To keep the intuitive visual representation of AHOI nets comprehensible, we assume the system to apply cleaning-up rules on temporary net structures after the activity that they were created for has been completed. Moreover, when simulating a system, the modeler should be able to grasp the system's state very quickly.

3.1 Skype User Clients as AHOI Components

Fig. 1 shows the AHOI component for the Skype client of a user "Alice". Each client component has the following basic structure:

[6] Skype is free to use and freely available at `http://www.skype.com`.

- One place typed by *State*, which is also named *State*, represents the current state of a Skype client. The data type *State* consists of the values *Offline*, *Online*, *SkypeMe*, and *DND*. The example client in Fig.1 is in state *Offline*.
- *SkypeName* places carry identities of Skype users, e.g. the token *Alice* on the place *User* indicates that Alice is the client's owner and the tokens on *Contacts* represent two Skype users she has in her contact list[7]. The place *WhitePages* is shared between all user clients and carries tokens corresponding to the tokens on the *User* places of all existing clients. A *SkypeName* token on place *CallRequest* would announce a request to the system for connection to the correspondent client, and similarly on place *ContactRequest* for exchange of contact data.
- *DataUnit* higher-order tokens, as shown in the left of Fig. 1, are a kind of agents that allow their owner client to send data to and receive data from other clients. The owner is indicated by the *SkypeName* token on the unit's place *Owner* and we denote a unit with owner X as *DU(X)*. The type *Data* may represent audible, textual, graphical etc. data, which a unit can generate, send, and receive by firing its transitions.
- If there is a token on the client's place *Ready2Talk* the client is supposed neither to be offline nor to participate in another call/conference, hence to be able to accept incoming calls.

An example firing step for activating the client has been discussed in Sect. 2.2. The remaining possible firing steps in Fig. 1 are changing the client's state to DoNotDisturb or SkypeMe by firing the corresponding transitions, deactivating the client by firing *deactivate*, which would delete the *DataUnit* token from *Ready2Talk*, or announcing requests for a either a connection to another client by firing *requestCall* or for contact exchange by firing *requestContact*. In the following section, we will see how the system reacts on requests by reconfiguring clients to allow more activities.

3.2 Request for Contact

In the following we suppose a minimal example of a system net that contains two client net components like the one shown in Fig. 1[8]. One has, as depicted, the tokens *Alice* on its *User* place, *Carol* on its *Contacts* place, and the object net token *DU[Alice]* on its *Template* place. The second is supposed to belong to a user named Bob, hence having the tokens *Bob* on its *User* place, *DU[Bob]* on its *Template* place and an empty *Contacts* place. Both clients are considered to be online, having the corresponding token on their *State* place, respectively.

[7] The data type *SkypeName* is some appropriate type for distinguishing identities, e.g. unique strings or integers.

[8] Note that these client components can be created in a system net (and be deleted) with reconfiguration rules dynamically! This realizes registration and resigning of Skype users. Basically, this rule consists of an empty left-hand side and the net in Fig. 1 as the right-hand side.

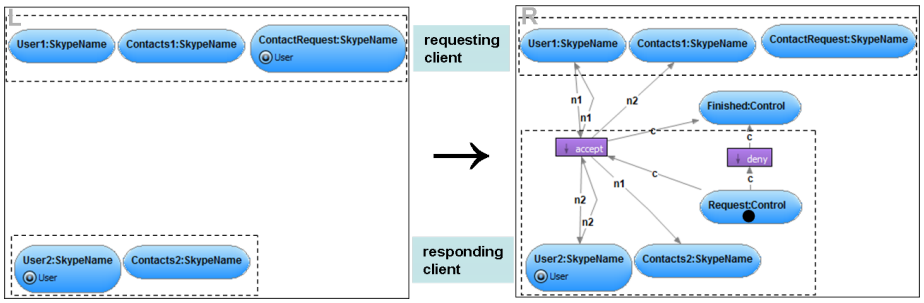

Fig. 3. Rule *CreateContactExchange* creating structure for contact exchange

Alice now wants to talk to Bob, which she would realize by firing *requestCall*, so that a token *Bob* is put on her *CallRequest* place. But she does not have his *SkypeName* token on her *Contacts* place, yet. So, before calling him she has to ask for his permission to add his contact to her contact list, represented by her *Contacts* place. This is according to the default procedure in Skype to follow.

To accomplish an exchange of user contacts, Alice fires her *requestContact* transition, so that the token *Bob* (assigned to the variable n in this firing step) is copied from the *WhitePages* place to Alice's *ContactRequest* place.

Remember, we want to separate strictly user behavior in the client from reactions by the system, following the modeling principles we stated at the beginning of this section. So, we let the Skype system react the Alice's request by applying the transformation rule *CreateContactExchange* shown in Fig. 3, extending the possible behavior of Bob's client.

In the left-hand side L of this rule we have three places that should be matched on the corresponding places of the requesting client and two places of the responding client[9]. Further, to be applicable, this rule needs a token value that can be assigned to the token variable *User*, one on *ContactRequest* and *User2* each. In our example scenario this would be the two tokens *Bob* indicating the user of Bob's client component and the request Alice just has announced before.

The effect of applying this rule is the creation of the structure in the right-hand side R and removing the request token on the place matched by *ContactRequest*. In the manipulated system net, the two clients are now connected with this structure and we interpret the newly created transitions as additional behavior for Bob's client. E.g., Bob can fire the transition *deny*, which moves a simple control token (assigned to variable c) to the *Finished* place and concludes the contact exchange request without further effect. Alternatively, he may fire *accept* which, besides moving the control token, will copy a token *Alice* to the Bob's *Contacts* place and a token *Bob* to Alice's *Contacts* place. This fol-

[9] The dashed frames and the boxes describing the roles in the middle of the rule are just a hint to understand to which component the matched places should be connected. They are actually not part of this or the following rules' syntax.

lows from having applied the rule matching *User1* place to Alice's *User* place, which surely carries a token *Alice* that, on firing *accept*, is assigned to variable *n1* and hence copied to Bob's *Contacts* place. The assignment of the token *Bob* (matched by the rule's token variable *User* before) to variable *n2* and copying it to Alice's *Contacts* place happens analogously.

In short, after Bob has fired *accept*, he now effectively has Alice on his contact list and vice versa and Alice is finally able to call him.

3.3 Cleaning the Model by Removing Dispensable Structures

After a request for contact exchange has been accepted or denied the additional structure created can be removed because none of the new transitions can fire any more, due to the control token being moved to *Finished*. We just can remove this now useless structure by the reversed creation rule, i.e. a rule that has basically *CreateContactExchange's* R as left-hand side and L as right-hand side. We don't carry this out in more detail[10] in this article, it just should demonstrate how reversed rules can be used in principle to keep the overall model lean and clear without confusing left-overs.

3.4 Creating Conferences

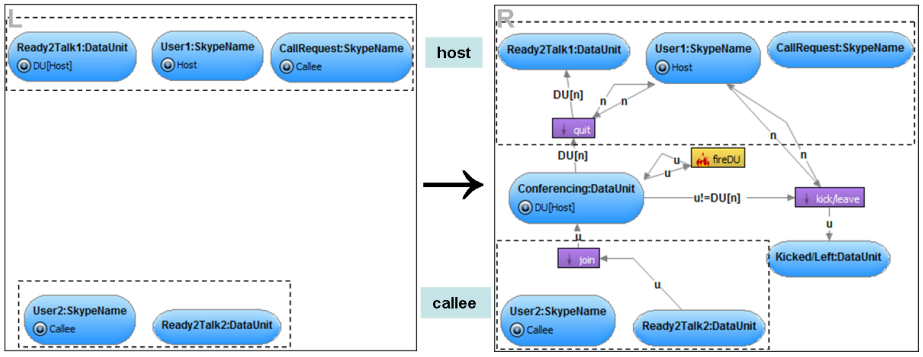

Fig. 4. Rule *CreateConference* creating a conference structure

Now, Alice wants to invite Bob to a direct call[11] and fires *requestCall* in her client component so that the token *Bob* that has been created before in

[10] Of course, the deleting rule is not supposed to delete a control token on the *Request* place and to create a *User* token as would do the simply reversed rule, but rather just to delete a control token on *Finished*.

[11] In Skype, you may invite additional contacts to a running conversation, so we consider a direct call just as an (initial) conference with two participants.

the contact exchange is copied to her *CallRequest* place. To allow the system to react to the request, we formulate the rule *CreateConference* depicted in Fig. 4. Similar to the previous rule, the four upper framed places in the left-hand side L should match the corresponding places of the conference host component, whereas the lower ones belong to the called client component. When applicated to our running example, the rule creates the conferencing structure in R, moves Alice's *DataUnit* token *DU[Alice]* from the *Ready2Talk* to the new *Conferencing* place, and deletes the request token *Bob* on *CallRequest*.

Being the host, Alice is attending the conference immediately after rule application; she is unavailable to other calls while her conference is running. Her only option is to *quit* (by firing the appropriate transition) and terminate the whole conference[12]. Bob may *join*, which would move his *DataUnit* token to *Conferencing* as well.

The conference is established now and we discuss the transitions *fireDU* and *kick/leave* in the following subsections.

3.5 Transmitting Data

We assume that Bob has joined the conference, so that his *DataUnit* token is now on the *Conferencing* place that just has been created by the rule *CreateConference*. Alice fired *send* in her *DataUnit* object net *DU[Alice]*[13], which copied the token *"Hello!"* from her unit's *Storage* to its *Out* place (cf. the left of Fig.1). We interpret a token on a *DataUnit's Out* place as a request to distribute the token value to all other *DataUnits* in the same conference.

The (vertically depicted) rule *Transmit* in Fig. 5 is a schema[14] whose instances each match a *DataUnit* token *sender* and a fixed number of receiver units to realize multicasting of data on a conference place; in our example we consider just Bob as a single receiver. In the right-hand side R, the matched tokens are replaced with algebraically calculated object nets, for which we use the following operations that we provide in the AHOI net's algebra:

- $out : DataUnit \to Data$ yields the token value on the *Out* place of the *DataUnit* passed as argument of this operation.
- $send : DataUnit \times Data \to DataUnit$ returns the passed *DataUnit* but removes the value of the *Data* argument from the unit's *Out* place.

[12] We assume the transition *quit* to have a firing condition, so that only *DU[Alice]* can be assigned to u. This is the reason why we need *quit* to be connected to Alice's *User* place; it needs to access her *SkypeName* token value.

[13] To fire object net transitions, we use the *fireDU* transition that has been created with the conference structure. It takes an object net token assigned to variable u, calculates algebraically the net that results after firing an enabled transition inside the object net u represents, and returns the fired net as a new object net token back to the conference place. For this we make use of the assumed operations on the Petri net token sort of AHOI nets. You may look at [8] for details of this construction.

[14] For now, we assume that we have a rule for each possible conference size, i.e. the number of participants. Recently, we proposed the more flexible and advanced approach of *Amalgamated Rules* for multicasting in [12].

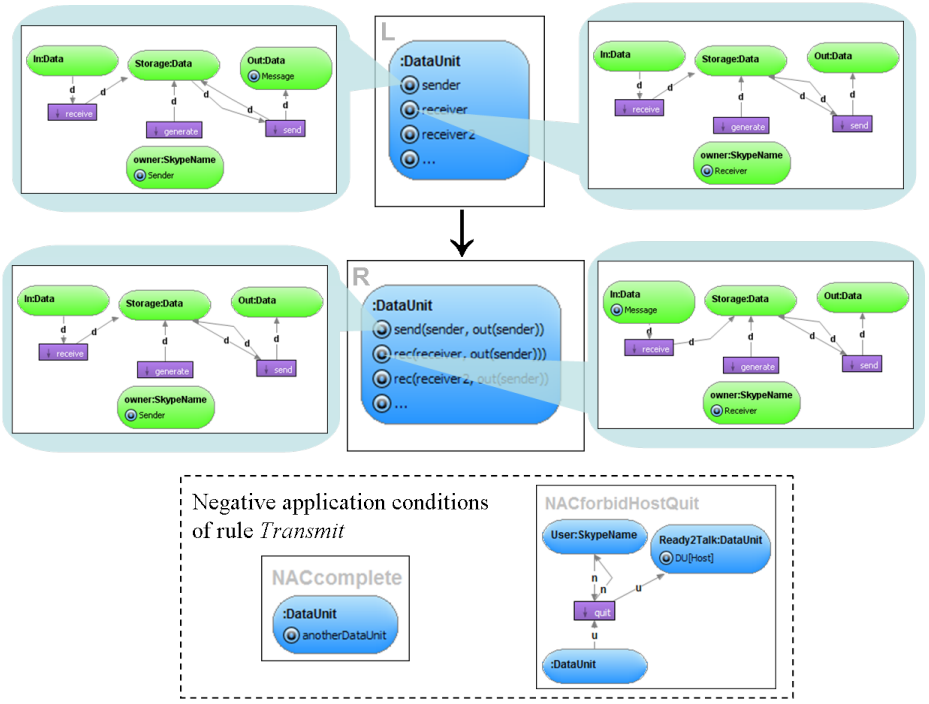

Fig. 5. Rule schema *Transmit* distributing data among *DataUnit* tokens

– $rec : DataUnit \times Data \rightarrow DataUnit$ is similar to operation *send* but returns the passed *DataUnit* where the *Data* argument value has been added to the unit's *In* place.

Now we can understand the terms in the right-hand side of *Transmit*: $send(sender, out(sender))$ is basically the *DataUnit sender* where the token has been removed from its *Out* place. Similarly, each term $rec(receiverX, out(sender))$ is the corresponding receiver *DataUnit* to which the *Data* token from the *sender* is added to. The green object nets besides the rule illustrate this. After applying this rule to our example, Bob can fire *receive* in his *DataUnit* to gather the transported *Data* from his input place *In* and the transmission has been completed.

To avoid incomplete transmissions (e.g. by applying a *Transmit* rule instance with just one receiver on a conference with three participants, hence with two receivers) and transmissions after the host has quit (which should immediately terminate the conference) we introduce *negative application conditions* (NACs) for rule *Transmit*. A rule is not applicable if there exists a valid occurrence for at least one of its NACs. With *NACcomplete* we ensure that the rule can be applied only if the left-hand side *L* matches all tokens on *Conferencing* so that there is not another unmatched *DataUnit* token left, allowing only complete transmissions to

all participants. *NACforbidHostQuit* prohibits a rule to be applied after the host has quit the conference (when his *DataUnit* can be located on his *Ready2Talk* place).

3.6 Joining and leaving Conferences

In Skype, the conference host may invite more participants to a running conference. In our example, Alice can fire another *callRequest* while she hosts a conference, so that the rule *InviteParticipant* in Fig. 6 can be applied. It simply connects another client to the conference like the intial rule for creating conferences before. Note that the invited participant does not necessarily have to be ready to talk to get invited, the rule rather ensures that Alice as the host has not quit the conference by requiring her *DataUnit* on the *Conferencing* place.

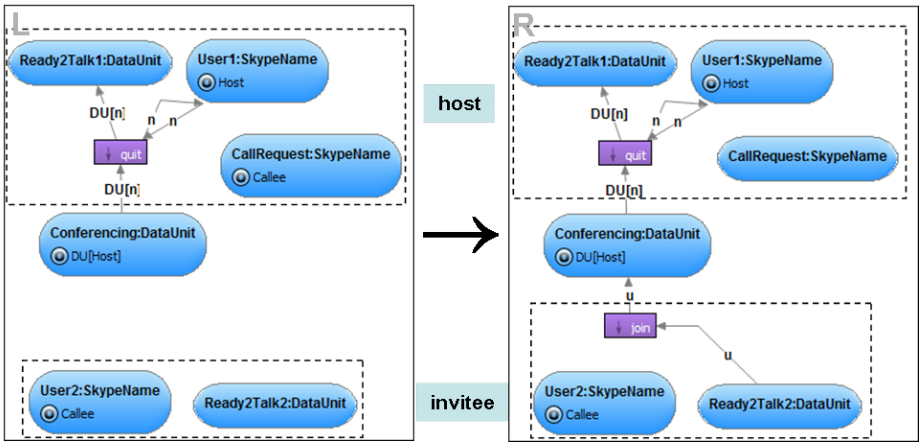

Fig. 6. Rule *InviteParticipant* connecting invited participants to conference

Every participant can leave the conference by firing *kick/leave*, but this can also be used by Alice to exclude participants from the conference. Note that Alice can only quit the conference via her own *quit* transition because the conference is considered to be finished when she does this. Other rules like *Transmit* respect that and do not allow to continue the conference so that the participants only option is to leave in this case. Again, a user firing *kick/leave* just announces a request that the system answers with application of rule *KickParticipant* in Fig. 7, which disconnects the client from the conference and moves its *DataUnit* token back to its *Ready2Talk* place.

It should be imaginable that an appropriate rule can be formulated to remove a conference whose participants all have left and whose host has quit, possibly with some NACs. This concludes the example scenario in a state where Alice's and Bob's clients have exchanged contacts and data and are now again unconnected client components.

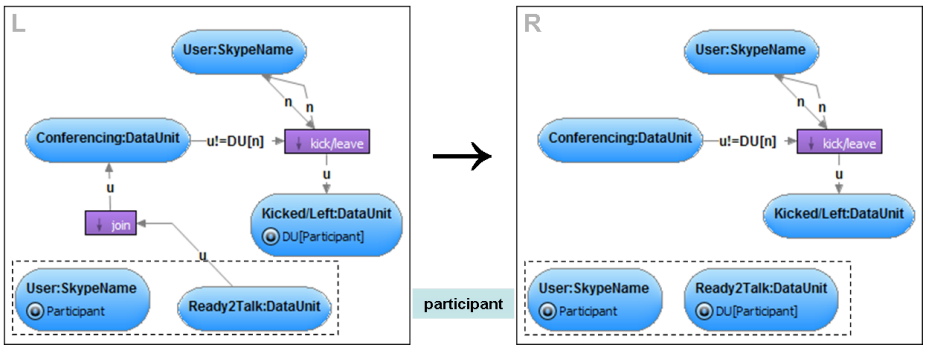

Fig. 7. Rule *KickParticipant* disconnecting participants who left a conference

4 Conclusion and Outlook

After introducing the general notion of Communication Spaces (CS), we discussed the new approach of AHOI nets as a formal modeling technique for CS. The examples, concerning particular features offered by Skype, show that this approach and the chosen modeling principles are powerful enough for first adequate modeling approach of CS. We have shown how we need to employ all the specialities of the AHOI nets as an integrated modeling technique:

- High-level data tokens are needed to represent data and user identities for effectively restricting communication.
- Moreover, higher-order tokens containing high-level nets are again used to represent users and their behavior and data in different system parts as conferences and chats [8].
- With reconfigurable Petri nets we enable the system to adapt itself to user requests and allow for a dynamic set of actors at system runtime. For more detail on the necessity of reconfiguration for modeling multicasting in dynamical Petri net systems we refer to [12].
- Petri nets with individual tokens [11] feature distinguishable tokens with identities, which are a technical premise to be able to formulate marking-changing rules. We have proven the existence of a weak adhesive high-level replacement system [6] for these nets, yielding a rich transformation theory.

To improve modeling usability, we plan to examine i.a. the following extensions of AHOI nets:

- An explicit control structure for defining rule sequences to ensure and enforce cleaning-up reconfigurations to be performed directly after the correspondent activity has terminated. Additionally, firing of transitions could be related to the application of particular rules and be comprised in the control structure.
- Amalgamated rule applications are similar to the "apply as long as possible" control structure, but in contrast, an amalgamated rule is applied to all

possible matches in the same step. With this technique, we can realize the schema of rule Transmit without explicitly formulating rules for all possible conference sizes. Moreover, we need this extension for non-local consistency-related concepts like Skype's Shared Groups, which are synchronized contact lists distributed between several clients. We contributed a first idea of how amalgamated rules can be used for multicasting data instead of the informal rule schema we used to formulate the rule *Transmit* in Fig. 5 [12].

Now that our formalism enables us to represent and simulate a model like Skype with the characterizing features of a CS, the most important part of our future work is to analyze and verify important properties of Skype in particular and to find/derive important properties we can prove for CS in general when using AHOI nets and the presented modeling approach with its principles in general. Examples of such properties are e.g. that a contact's owner must have confirmed all correspondent entries in his contact list, or that the system respects the clients' privacy settings when establishing communication via reconfiguration. To verify such properties we intend to make use of the rich theory of (high-level) Petri nets [4] and high-level replacement systems [6], especially the result for consistency and independency. We are currently elaborating the transformation theory of nets with individual tokens [11] by lifting the results of the collective approach and developing new analysis concepts.

To provide tool support for modeling and analyzing, we are also extending a graphical editor as an Eclipse plugin for a simplified kind of the AHL nets in [8] to support the AHOI formalism as we introduced it in this article.

References

1. Agha, G.: ACTORS: A Model of Concurrent Computation in Distributed Systems. PhD thesis, MIT (1985) Cambridge: MIT Press.
2. Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press (June 1999)
3. Reisig, W.: Petri Nets: An Introduction. Volume 4 of EATCS Monographs on Theoretical Computer Science. (1985)
4. Jensen, K., Rozenberg, G., eds.: High-Level Petri Nets. (1991)
5. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1: Equations and Initial Semantics. Volume 6 of EATCS Monographs on Theoretical Computer Science., Berlin (1985)
6. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. EATCS Monographs in Theoretical Computer Science. (2006)
7. Ehrig, H., Padberg, J., Ribeiro, L.: Algebraic high-level nets: Petri nets revisited. In: Proc. of the ADT-COMPASS Workshop'92 (Caldes de Malavella, Spain), Berlin (1993) Technical Report TUB93-06.
8. Hoffmann, K., Mossakowski, T., Ehrig, H.: 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. Volume 3536 of LNCS. Springer Verlag (2005) 268–288
9. Valk, R.: Concurrency in communicating object petri nets. In: Concurrent Object-Oriented Programming and Petri Nets. (2001) 164–195

10. Ehrig, H., Hoffmann, K., Padberg, J., Ermel, C., Prange, U., Biermann, E., Modica, T.: Petri Net Transformations. In: Petri Net Theory and Applications. I-Tech Education and Publication (2008) 1–16
11. Ehrig, H., Modica, T., Biermann, E., Ermel, C., Hoffmann, K., Prange, U.: Low- and high-level petri nets with individual tokens. Technical Report 13/2009, Fakultät IV - Technische Unversität Berlin (2009) to appear.
12. Biermann, E., Ehrig, H., Ermel, C., Hoffmann, K., Modica, T.: Modeling Multicasting in Dynamic Communication-based Systems by Reconfigurable High-level Petri Nets. In: 2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL-HCC'09). (2009)