Characterizing the Tool-notation-people *Triplet* in Software Modeling Tasks

Juliana Jansen Ferreira  
SERG/Informática – PUC-Rio  
R. Marquês de São Vicente 225  
Rio de Janeiro – RJ, Brasil  
jferreira@inf.puc-rio.br

Clarisse Sieckenius de Souza  
SERG/Informática – PUC-Rio  
R. Marquês de São Vicente 225  
Rio de Janeiro – RJ, Brasil  
clarisse@inf.puc-rio.br

Renato Cerqueira  
IBM Research Brazil  
Avenida Pasteur 138  
Rio de Janeiro - RJ, Brasil  
rcerq@br.ibm.com

**ABSTRACT**

The existence of some kind of relationship between the usability of software development tools and the quality of use of the development process end product would not be surprising. Yet, this topic hasn’t attracted a substantial share of interest among HCI or Software Engineering researchers, possibly because we lack the appropriate conceptual and technical tools to address the problem. In this paper, we articulate a tool-notation-people *triplet* and suggest that it can be used in interaction design evaluation of modeling tools used by many software developers. The evaluation is carried out with a specific method that combines cognitive and communicative dimensions of such tools and characterizes how the tool-notation-people *triplet* is instantiated for the case under examination. We demonstrate the value of our proposal with a study of IBM RSA, a popular software modeling tool. The interest of this work for the HCI community is to provide a set of resourceful tools – combined method and triplet – to identify and collect interaction issues that could be used to improve the design of modeling support tools.

**Author Keywords**

Human-computer interaction; software development tools; notations; cognitive dimensions of notations; usability; communicability; semiotic engineering

**ACM Classification Keywords**

H.5.2 User Interfaces: Evaluation/methodology. H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

**INTRODUCTION**

Software development is heavily supported by tools throughout the entire process. From the design phase through the testing phase, various tools are used to support people in producing intermediary artifacts as well as the final achieved software piece. The quality of development tools can impact the quality of produced software in a number of ways, including the end user’s interaction experience. What we don’t know is the potential magnitude of this impact, where and how it originates, and finally how negative impacts can be avoided (probably by improving the quality of development tools themselves). The challenge for researchers who want to know the answers to these questions is that software development tools – and modeling tools in particular – are very complex, hard to design and hard to evaluate. [3]

Our long-range research goal is to investigate if some of the blunders that end-users experience while interacting with computer technologies are related to blunders that software designers have experienced themselves while using software development tools to produce such technologies. The first step in the long way towards this goal is to identify if, how, when, where and why interactive blunders happen while software designers are acting as users of software development tools.

In all software engineering projects, modeling is an important activity at some point along the process. [18] From conceptual modeling to database definition, models are built and used in all major software development phases. Modeling tasks must take into consideration a number of factors that cannot be isolated from each other. [20] Most professional models are built with tool support. Therefore, tools highly influence how models are composed. Models, in turn, are expressed by notations. OMG\(^1\) for example, is responsible for specifying the most widely used software development modeling notations, the Unified Modeling Language (UML). [15] As a result, modeling tool designers need to be aware of notation specifications, definitions and updates in order to provide proper support for modeling tasks. Last but not least, models are built by people and usually also for people.

We frame these three factors together into what we call the *tool-notation-people triplet*. We propose that this *triplet* can be studied and elaborated with the aid of HCI (Human-Computer Interaction) methods and techniques commonly

---

Inspection methods are lower-cost evaluation aids (compared to user observation methods) and can focus on different aspects of interaction. Here, they have been used to address two specific aspects of interaction affecting the triplet: usability and communicability. Usability is a notion strongly rooted in cognitive HCI perspectives, whereas communicability originates from a semiotic approach.

In this paper we present research combining usability- and communicability-driven inspections to evaluate HCI design aspects of a specific modeling tool. The result of the evaluation offers a rich characterization of the triplet, which allows us to infer potential consequences of developers’ interaction with the tool in the broader context of software development. (Fig. 1)

MODELS IN SOFTWARE DEVELOPMENT

In the software development process, models can be built with different purposes. For example, they can be built as documentation artifacts, a static representation of (part of) the system at a given moment in time. They can also work as a means of communication among people involved in the software development process. [17] As such, they may achieve different goals: 1) to work as a message in itself, sent from model creators to model users; or 2) to work as a representation of ongoing collaborative interaction, supporting discussions where the represented model serves as an orienting context.

Models as means of communication constitute the object of our research. One key point to consider about models in software development is that they are built by people (an individual or a group) and meant for people. Therefore, a model carries in it its creators’ understanding, intent and choices. Model users, down the line in the development process, will pick up on these meanings and, in turn, develop their own understanding, intent and choices regarding the software system that is under construction. Thus, modeling tools and notations play a key role in helping people (model creators) design, build and send their ‘message’ to other people (model users). Eventually, as models are reified in the form of technology, the entire set of meanings assigned to them throughout the development process will determine how users can or must interact with the system.

Although the importance of tool support for software modeling activities is widely recognized, there are more technical and social factors to consider in this scenario than has been usually the case. [20] Most of the research about modeling tools has focused on the relation between tools and supported notations. [3][12] UML [15] has been the main focus of evaluation and investigation, [17][19] mainly because it is extensively used in spite of some known problems and limitations. In general, researchers haven’t been looking at how tools and supported notations affect...
the way people create and use models – especially models that refer and relate to each other – in a broader perspective that includes technical factors like usability and communicability of modeling tool interfaces. The latter, in particular, is strongly related to the social factors in the process that have deserved little attention so far.

**USABILITY-DRIVEN AND COMMUNICABILITY-DRIVEN INSPECTIONS OF MODELING TOOL INTERFACES**

Our research is based on two HCI perspectives that have comparable depth, but significant contrasts: Cognitive Engineering [14] and Semiotic Engineering. [7]

Cognitive engineering, the usability-driven perspective and one of the best known theoretical characterizations of human-computer interaction, has strongly influenced the User-Centered Systems Design approach. It focuses on user-system interaction. When defining the central phenomenon of interest in HCI, this theory – as its name says – centers around users alone, not taking into consideration any of the profile of many other people (e.g. systems designers and developers, systems owners, etc.) who, in one way or another, may influence the way users interact with computers.

Semiotic Engineering, the communicability-driven perspective that we adopted, emphasizes communication and significiation processes taking place in interaction, and brings HCI designers onto the stage of human-computer interaction. Systems’ interfaces convey their creators’ and producers’ message to users. [7] They function as the designer’s proxy (deputy) during interaction, thus achieving mediated designer-user communication.

The most relevant difference between these HCI perspectives is which people each theory accounts for. The usability-driven perspective focuses on users and their experience while interacting with the system interface. Although it obviously acknowledges that systems are built by people (especially HCI designers), Cognitive Engineering keeps systems producers out of the picture when examining interaction. The communicability-driven perspective, however, views systems builders (especially HCI designers) as part of the focal interactive phenomenon to be accounted for.

The combination of usability-driven and communicability-driven perspectives enables a more comprehensive inspection of all relevant parts of the object of analysis and the relations between them. We selected CDNf [2] as the usability-driven inspection tool, and MetacommmT as the communicability-driven inspection method for our study. The latter is actually a basic common step in two much more elaborated methods proposed by Semiotic Engineering, namely the Semiotic Inspection Method [4][5] and the Communicability Evaluation Method. [5][6]

CDNf and MetacommmT are presented in further detail in the following subsections.

**Usability-driven perspective inspection**

CDNf is a prime candidate for evaluating the usability of tools dealing with notations. It defines a set of design principles for creating or evaluating notations, user interfaces and programming languages used with information artifacts. This framework provides a common vocabulary for discussing many cognitive factors of such representation-building systems. Its aim is to improve the quality of discussions and decisions in design and evaluation activity. [2]

There are fourteen dimensions in CDNf. Just for the sake of a very brief illustration, one of them is Secondary notation. It refers to the situation when the notation allows for the inclusion of extra information in means other than formal syntax, which is a way of expanding the notation. Another dimension is Error-proneness, which refers to the situation when the notation leads to mistakes and the system (where the notation is used) gives little protection against them. Other dimensions with self-explanatory names are, for instance, Hard mental operations, Abstraction and Visibility. Detailed descriptions of all dimensions and how to use them in HCI evaluation (or design) can be found elsewhere. [2][11][16]

The evaluation of a notation system design with CDNf has five steps: (1) get to know the system; (2) decide what the user will be doing with the notation; (3) choose some representative tasks; (4) for each step in each task, ask whether the user can choose where to start; how a mistake will be corrected; what if there are second thoughts; what abstractions are being used; and so on, for the other dimensions. These steps will generate the notation’s observed profile. The final step (5) is to compare the observed profile with the ideal notation profile for the selected type(s) of activity.

CDNf is a particularly powerful framework for evaluating tools where the use of notation is pervasive and central. It has been applied in different contexts like programming languages [11] and graphical user interfaces [2]. Its focus on notation turns CDNf into a suitable inspection method to handle the “N” part of the triplet.

**Communicability-driven perspective inspection**

Semiotic Engineering defines a template to characterize key meaning elements of the designers’ message conveyed to users through systems interfaces. [7] In essence, the message in this template (MetacommmT) communicates to users how they can or must communicate with the system in order to achieve various kinds of goals and effects (hence the notion of metacommunication, a communication about

---

3 Semiotic Engineering uses the term ‘designer’ or ‘designers’ to refer to the group of people who conceive and produce an interactive system. Therefore, it should be taken to mean this entire group collectively, rather than one or other particular group member.
communication). MetacommT is used to guide communicability investigation, from either a sender’s [4] or a receiver’s [6] perspective, by providing an abstract and logically articulated representation of the designers’ discourse about their understanding of who the users are (what needs and preferences they have, what may be their goals and expectations, their abilities, their knowledge, etc.) as well as the design decisions and choices that ultimately express their design intent (and its expected value for the targeted users) [5]. The content and structure of MetacommT is the following:

“Here is my understanding of who you are, what I’ve learned you want or need to do, in which preferred ways, and why. This is the system that I have, therefore, designed for you, and this is the way you can or should use it in order to fulfill a range of purposes that fall within this vision.” [7]

The first person of discourse in the template (referred to as “I”, “my”) stands for the designer, and the second (referred to as “you”, “your”) for the user. The third person in discourse (referred to as “it”, “this”) stands for the system, represented by its interface. Together, they characterize the participants in an elaborate metacommunication process that takes place during human-computer interaction.

In the Semiotic Inspection Method (SIM) [4], MetacommT orients HCI evaluators while preparing inspection scenarios. In this phase, they must carry out an informal walkthrough of the chosen artifact, aiming at establishing the focus of analysis. Evaluators must thus determine the minimal elements and conditions to inspect communication processes, that is, the senders, receivers, message contents, message codes, communication channels and context. Additionally – and critically important for the analysis – communication must be able to achieve the users’ goals. Therefore, in communicability evaluations carried out with either an inspection method [4] or a user observation method [6], evaluators must begin by identifying: (i) who are the intended users of the system; and (ii) what are the top-level goals and activities that the system supports.

The filling out of MetacommT in the research presented in this paper provides critical information about the template’s first portion (“who you are”) and the last one (“range of purposes that fall within this vision.”), which are not usually taken into account by other inspection methods commonly used in HCI. [13] And it also provides information about the middle part of the template, as other HCI inspection methods (including CDNf) do.

MetacommT deals with all the elements of the triplet, but its communication perspective focuses more strongly on the “P” part of the triplet. The template represents the message from designers (people) to users (people) through the software interface (tool). The software interface is composed by some notation chosen by the designers to deliver their message efficiently and effectively. In the specific context of our present research about tools used to build software models, the modeling notation is an important subset of the entire notation system used by the designers of modeling tools (other notations include interface control symbols, the representations of system states, and the like).

COMBINED USABILITY AND COMMUNICABILITY-DRIVEN EVALUATION

As already mentioned, the long-range goal of our research is to be able to explore the possibility that some of the blunders that end users experience while interacting with computer technologies are related to blunders that designers experience themselves while using software development tools. To this end, at this stage of our research, we inspect the usability and communicability of modeling tools. The result of the inspection – as is the case with typical inspections of interactive technology in HCI – is a characterization of issues that can negatively affect the experience of tool users.

We articulate the triplet to aid the combined HCI inspections as a reference of the context in which the modeling tool will be inspected. Before the inspection, the “T” part of the triplet is not fully characterized; we only know which tool will be inspected. The “N” and “P” parts are only loosely specified in general terms (e. g. “N” is UML, “P” represents the requirement engineers). This is the triplet baseline for the inspection. At the end of the combined method inspection, we will have learned more about the “T” part, and thus we can completely characterize the triplet for the specific inspection scenario under study.

The characterized triplet will indicate existing usability and communicability issues in the analyzed tool and point at the source of the problem. For example, one issue may be related to the notation alone, others may be related to people’s roles in ongoing software development communications, the relation between notation and tool, the relation between people and tool, or any other element or relation in the triplet.

Combining usability and communicability perspectives into a single method is justified by the fact that the artifacts produced by software development tools – in particular by modeling tools – often (if not always) serve the purpose of “communicating” in and of itself the understanding and the intent of team members who have produced artifacts (models) for other team members to use or evolve. A combined method allows us to inspect how easily these artifacts can be built and understood, as well as how easily they can speak for their creators in an ongoing process of elaboration and development.

Step 1  We start by defining the study scenario. In this step, we select the tool for the inspection, the task(s) to be performed, and the general user profile. This constitutes the triplet baseline of the study. Note that because of the semiotic perspective of our method, the PEOPLE portion in
the baseline triplet must also include the designers of the
selected tool.

Step 2 We proceed with a cognitive analysis of notations,
which amounts to a standard application of CDNf [2] and
the resulting instantiation of some of the aspects in the
triplet.

Step 3 The next step is a communicative analysis, which
consists of filling out Metacommt as we inspect the
behavior of the tool in the selected scenario. Because the
communicative aspects are inspected after cognitive
aspects, we take into account the results of the cognitive
analysis carried out in the previous step. This is an
important feature of the method, since it explains and
justifies why we call it a combined method. Notice that
steps 2 and 3 are thus not independent from each other. Step
3 benefits from the results obtained at step 2, and the
process of analysis is thus incremental.

Step 4 The final step of our method is to evaluate the
consistency among the features and relations emerging
from the Metacommt fill out. In it, as already mentioned,
the structure and logic of communication are informed by
the semiotic analysis and the mental activity carried out by
people while using the tool is informed by the cognitive
analysis. The conclusion of the method is an enriched
characterization of how the inspected tool supports users in
activities that are directly or indirectly related with the
inspection scenario. This is tantamount to completing an
entire characterization of the triplet.

An illustration of the method in a study of IBM RSA
To illustrate the method, we carried out an inspection of a
well-known software modeling tool, namely IBM Rational
System Architect (IBM RSA)\(^4\). The complete inspection
data is presented in a technical report\(^5\). This tool was
selected because it features as one of the top modeling tools
in the Gartner Magic Quadrant of Enterprise Architecture
tools\(^6\) used by the software engineering industry.

Our intention with the study was to investigate how IBM
RSA supports software modelers. The chosen task was to
build a UML activity model for a vacation request process,
which is usually part of human resources management
systems in large companies. The activity model should
describe a flow that features a vacation request by
employee, the manager’s approval, the system notification
and other subsidiary activities. The activity model is a UML
behavior diagram that shows the flow of control or object

flow with emphasis on the sequence and conditions of the
flow. Activity models are intended to model both
computational and organizational processes. They play the
role of an “interface model” between the business and the
technology portions of the system being developed. It can
be used to model more abstract activities (like interactions
between system and users or between systems) or to model
specific object parts of the system itself, showing how they
change along the process and over time. [15] This interface
role of activity models was the reason for selecting them for
our study.

The user profile for our scenario, which seems to be a
plausible situation in software development settings, was
that of an active professional in business modeling,
requirements modeling and conceptual modeling. However,
we assumed that this user had never used IBM RSA and
had little experience with UML activity models in real
projects. All he knew about them was what he had learned
in professional training and education some time ago.

In sum, for this study, our baseline triplet features IBM
RSA as “T”, UML activity model notation as “N” and an
active professional modeler that doesn’t have experience
with IBM RSA and knows of UML activity models only in
theory as “P”.

Usability-Cognitive (CDNF) analysis
By using the established scenario and baseline triplet, we
followed all the steps proposed by CDNF (see previous
section) and found a number of usability issues promoted
by the notations that appear in IBM RSA’s interface. For
lack of space, and given our purpose of illustrating the
method (and not making a detailed usability analysis of this
modeling tool), we concentrate on the main findings that
the inspector generates in the fourth step of the CDNF
method.

In this step we identified interesting issues related to how
UML activity model elements can be connected to each
other in IBM RSA. By comparison with the “ideal profile”
specified in the fifth step of the CDNF inspection, we
arrived at the kinds of findings discussed below.

\(^4\) IBM Rational System Architect (IBM RSA) trial 8.0.4,

\(^5\) http://www2.serg.inf.puc-rio.br/index.php/published-
work/304-combining-semiotic-and-cognitive-perspectives-
to-evaluate-software-interaction-design

\(^6\) Gartner Group - https://www.gartner.com/doc/2601526

Fig. 2. A decision node associated with itself at IBM RSA
In Fig. 2, we show that in RSA an activity model can be
represented with a ‘decision’ node connected to itself. This
IHC 2014 Proceedings - Full Paper

notational configuration refers to a loop that can never be solved, because there isn’t any action to define where the activity flow will go after the ‘decision’ is made. The same kind of problematic situation is verified when an ‘initial’ node can be directly connected to a ‘final’ node (Fig. 3). This association cannot mean a proper model configuration, since no activity or action is assigned to the system (it doesn’t act in any way). Although this feature relates to an important cognitive dimension in usability terms, namely Progressive evaluation (the ability to produce incomplete representations that are incrementally composed along the modeling process, in our case), it does so in such a way that a usability problem is entailed by the interaction designers’ solution. Because there is no distinction between provisional representations to be further specified and definitive representations that express the modeler’s final decision, this IBM RSA feature is also associated with another cognitive dimension of notations, namely Error-proneness. The notation invites mistakes and the system gives little protection against them. Using the triplet to characterize this issue, we can say that the “T” allows the user to build models using the “N”, but the model built may not have a proper meaning as far as the UML activity notation semantics is concerned. The user building the model (“P”) can be misled by “T”, while using the “N”.

This laissez faire strategy is however not consistently adopted by IBM RSA. In other situations, the tool does not allow users to make meaningless associations between elements, respecting constraints of UML specification. For example, an incoming flow cannot be associated with an ‘initial’ element. Yet, the feedback given to the user in such problematic situation may be too quick or hard to perceive. A ‘prohibited’ sign quickly flashes when the user tries to make an incorrect association between elements (Fig. 3). If the user is not paying close attention, or is moving the mouse rapidly across the interface display, the feedback about the mistake may not be noticed. As a result, he may be left wondering why the association is not being done. Among possible causes, he may give more priority to a problem with the tool than a problem with his model.

![Fig. 3. Initial and Final node](image)

This characteristic is related to two cognitive dimensions in CDNf. One, of course, is Visibility (the visual feedback is not visible enough). The other is Diffuseness, which refers to how many symbols (or how large a representational space) the notation requires to express meaningful content. The user has just one symbol, a quick indication that he is trying to do something wrong, nothing else (e.g. no redundancy or additional symbolic structures). In terms of the triplet, feedback representation, “T” does not provide proper information to “P” that “N” is being mistakenly used. This is a very common usability problem in a number of applications, not only modeling tools.

![Fig. 4. Decision and Merge nodes at IBM RSA](image)

Other kinds of cognitive barriers are illustrated in Fig. 4. There we can see two different elements with the same visual representation: merge and decision node. Although this is a problem with UML, which assigns the same representation to these two elements, it is curious to see that a modeling tool (designed to support modelers) should decline the opportunity to call the users’ attention to the problem visually. According to UML, the behavior of ‘Decision’ and ‘Merge’ nodes is different in that only one flow can arrive at a decision node and only one flow can come out of a merge node. As already anticipated in the discussion of the problem illustrated by Fig. 3, IBM RSA helps the user by not letting him exceed the limit of flows. A potentially serious problem, however, arises from the way how IBM RSA designers decided to take action when a mistake is detected. If the user tries to connect a second flow into a decision node, IBM RSA automatically creates a flow out of the node, without further notice. The user may thus get the impression that he mistakenly used an out flow when he should have used an in flow, and try again. It may thus take a while for the user to realize what is happening.

This problem is related to another cognitive dimension of notations, Hard mental operation (a high demand on cognitive resources for the user to understand the notation system). In terms of the triplet, this issue affects all three elements, as well as the relations among them. By preventing the user from making an incorrect connection in a UML activity model, IBM RSA helps the modeler to achieve better results (“T-N-P”). It also tells us something about “T-P”, as previously discussed in the feedback example, as well as about “N” per se (the ambiguity in UML notation). Because it provides a description of the element once it is selected, IBM-RSA also supports the “T-N” relation, clarifying the meaning of the notation for the people who use it.

Finally, in Fig. 5 we can see that IBM RSA allows us to build a visually valid activity model without any activity node in it. The UML specification [15] states that an activity model must have at least one activity node. However, IBM RSA allows the user to save that model as is without any warning. More importantly, the tool provides a
“UML model validation” function that does not indicate any problem with the model, as shown in the encircled area of the image in Fig. 5.

Fig. 5. Activity model without activity

This is again a problem referring to Error-proneness, which has been mentioned in previous paragraphs. In terms of our triplet, “T” is not handling “N” properly and may mislead “P” into building an activity model using the specifications of “N” that does not make any sense to other “P” in the process. Although this is an extreme case of wrong modeling, which is not rigorously compatible with the profile of the user that guides our scenario (who, after all, would build an activity model without activities?), it is nevertheless remarkable that IBM RSA oscillates between verifying and not verifying the semantics of the model under construction. This in itself creates error proneness (will the modelers ever know if they can trust the verification or not?) and this is our justification to include this last example as a valid situation in our inspection.

The overall result of CDNf analysis underlines discrepancies in the behavior of IBM RSA regarding notation semantics. Sometimes its behavior is consistent with UML notation semantic, preventing or correcting wrong associations. We can say that in this case IBM RSA properly behaves as a modeling tool. But sometimes it does not verify semantic inconsistencies in the use of UML notation. In this case we can say that it behaves like a drawing tool.

CDNf therefore helps us describe usability issues with connecting UML activity model elements with a rich vocabulary and insightful perspective. However, although we could elaborate an in-depth understanding of the challenges faced by modelers as they tried to build a specific activity model, we did not have a systematic way of inferring the consequences of these challenges in the broader context of how software models are used in the development process. This is an elaboration of the “P” node in our triplet, along with its relations with other elements, which is analyzed in the next step of our combined method.

Incremental communicability-driven (MetacommT) analysis

This step consists of filling out the MetacommT starting from what we learned during the CDNf analysis and then completing the gaps with what can be learned with an inspection of metacommunication aspects evidenced by the tool.

CDNf deals, in a very structured way, with the middle part of the template: “what I’ve learned you want or need to do, in which preferred ways, and why. [...] this is the way you can or should use it...” This is so because CDNf concentrates on the cognitive processes that take place as users interact with systems. These processes involve goals, interpretations and abundant use of notations, as was shown in the paragraphs above. However, because like all other usability-driven approaches CDNf does not contemplate the designers’ intent and how the final artifact communicates such intent to users, there are important gaps in MetacommT. They are mainly located in the parts of the template where the first person of discourse (“I”, “my”) reveals the presence of the system designer in user-system interaction. We are talking about the following portions of the template: “Here is my understanding of who you are, ... This is the system that I have therefore designed for you, ... in order to fulfill a range of purposes that fall within this vision.”

The parts of the MetacommT where the designer expresses his intent and vision can help identify further issues (or consequences of identified issues) while interacting with IBM RSA. We start with a key observation resulting from CDNf that this tool sometimes behaves like a modeling tool, and some other times like a drawing tool. In terms of communication, this hybrid behavior requires that users be aware of this oscillation while creating their models. However, when we look at MetacommT, we realize that IBM RSA designers must have a broader understanding of why users are creating models in the first place. As already mentioned in previous sections, most of the time models are meant for (other) people to use throughout the software development process. Therefore, IBM RSA designers must contemplate the fact that model representations are read (and not only composed, whether as a model or a drawing). The reading, however, is deeply affected by the reader’s assumptions about the tool with which the model was created. For example, if the reader thinks that IBM RSA is a modeling tool, which checks the semantic correctness of the model, then this reader will assign certain qualities to the model that will not necessarily be assigned if he believes that IBM RSA is a drawing tool.

Consequently, when filling out MetacommT, this aspect becomes apparent.

“Here is my understanding of who you are...” While inspecting IBM RSA to fill out the first part of MetacommT, we came across help content that explicitly defines who the intended users are. It says that this tool supports UML notation, but that if users need more detailed information about the notation, they should see the OMG specification. In other words, IBM RSA designers somehow excuse themselves for not providing explanations further information about UML. Therefore, we can assume that in their view this is not a critical need of the users (or
else they should have included this matter in their tool). The way we fill out MetacommT thus begins with:

“I understand that you have a fair amount of experience with modeling and the UML activity model notation...”

The MetacommT adds the user profile perspective from the designer’s message to the combined evaluation. Furthermore, with the MetacommT we can reconstruct the message sent from the designer (people) to the user (people), so we bring the designer’s role into the evaluation of IBM RSA communicability. This is also not possible with the cognitive perspectives that focus only on the user. The “P” part of the triplet now considers the designer as well as the user of the tool.

“...what I’ve learned you want or need to do, in which preferred ways, and why.” Given the behavior of IBM RSA illustrated in preceding portions of this paper, we can fill out the next part of MetacommT with the following:

“... you need to build activity models with some help to prevent incorrect connections. For example, you are likely to be confused with ‘Decision’ and ‘Merge’ nodes, but you can spot obvious modeling mistakes by yourself (like producing a model without a single activity, creating endless decision loops, etc.). You may wish to build models incrementally, which means that some intermediary stages in the process may be obviously wrong from a UML semantics perspective. We therefore don’t check the semantics of the model all the time, and trust you to be able to track where the evolving model needs to be elaborated further in order to be semantically correct ...

As we proceed to the next portions of the template, we incorporate observations that stem from our own communicative inspection of the tool.

“...This is the system that I have therefore designed for you, and this is the way you can or should use it...”. The prevalence of direct manipulation in the interface suggests that the designers’ message to users includes the following:

“... by connecting UML activity elements using the mouse to build the model and the keyboard to assign a name to each element, you can build a visual representation of the model and invoke a semantic verification function to check the correctness of your model. The semantic verification is not complete, however. Only a subset of connections can be checked, and you will find this by trial and error. ...

The latter sentence in the above filling out of MetacommT is communicated by the fact that there is no visual difference in a model between semantically verifiable portions of it and semantically non-verifiable ones. The consequences of this design vision now begin to emerge very clearly. IBM RSA designers, as is the case of other modeling tool designers, typically limit their perspective on users to a very narrow context of modeling activities. In it, the fact that other people are going to read models produced by their tool is usually out of sight. Therefore, not only does the tool not support model reading, but it is also prone to introducing mistakes in the model that are going to propagate indefinitely in subsequent stages of software development.

“...in order to fulfill a range of purposes that fall within this vision.” This last part of MetacommT makes the points raised above quite explicit. It could be filled out in the following way:

“... in order to build UML activity models that you can store, print, publish, revise, transform into code or discard along the software development process.”

Note the indirect mention to other “P” in the development team when the designers communicate that models can be printed, published (for sharing purposes) or transformed into code, but the apparent inconsistence between this and the fact that the final model representation does not clearly communicate which parts are semantically verified, and which parts aren’t (or cannot be, and why).

The complete filling out of MetacommT at the end of the combined evaluation method characterizes both the communicability of the tool under study and its usability profile as produced by CDNf. In the illustrated examples above, we see that the message communicated through the interface contains a number of questionable design choices – no matter if they have been made consciously by the designers, or if they are an unsuspected consequence of other design choices. They affect model users one way or another. Thus, it provides an in depth and in detail characterization of the triplet for the selected scenario. It shows all parts in the inspected context with tool, notation and user profiles. It also highlights the tool designers’ role and contribution as part of the “P” node in the triplet, something that is usually not taken into account although, as we show, it can deeply affect our perception of the interactive quality of modeling tools.

**Conclusion about the combined analysis**

This step consists of presenting the conclusions about the combined analysis, how one analysis complemented the other and what the evaluator was able to learn from the analysis.

We can compare the triplet baseline with the complete triplet characterization and verify if all nodes and relations are finally consistent with each other.

For lack of space we will abbreviate the illustration of conclusions, skipping the details and concentrating on what we think is the fact with the strongest potential impact for the software development process and – possibly – for the end users of the system being developed.

The hybrid behavior of IBM RSA can lead its users into conceptual confusion. What kind of “model” is does this tool actually help them build? Is the final representation a semantically correct model or just a drawing? Whose
responsibility is it to verify correctness? What does it mean if the tool says that a given model “has no errors”? How should model readers interpret the final representation? How much does the quality of the represented model depend on the tool? How much of it depends on the modeler? Are tool-supported models better than manually built ones? How and Why?

An interesting part of concluding our analysis is to realize that similar questions are asked by the software engineering community without taking tool usability and tool communicability into account. [16][17][20] But, as we hope our illustration has shown, if tool designers become more aware of what they are doing when designing systems like IBM RSA and many others, a number of computer-supported modeling issues can be alleviated. Take for example the problem with partial semantic verification. If semantically verified parts are communicated with different signs than non-verified parts, this is already a considerable improvement independently of extensions in the semantic verification scope. And it is a relatively simple improvement in terms of HCI.

Another interesting item in our conclusion is the fact that model-building tools incorporate a number of reading features that are necessary for model-editing. A small shift in perspective – thinking about somebody else reading a model – could easily provide a number of usability and communicability requirements that, if attended to, might improve modeling tool designs in considerable ways. This kind of reflection is directly prompted by portions of MetacommT, especially the last one (as we have shown).

**DISCUSSION ABOUT THE COMBINED EVALUATION OF MODELING TOOLS**

The combination of usability- and communicability-driven HCI inspections of interaction with modeling tools produce complementary characteristics that account for all nodes and relations in the triplet we propose for evaluating and improving the design of modeling support tools. The TNP triplet is a new resource for research, associated with the combined method already used on previous papers. [1][9][10].

In itself, the triplet is a guide to organize findings obtained with the combined inspection. For each one of the findings, we look at all other parts of the triplet as well as the relations between them in order to see what implications such findings may have for the tool user, individually, and for the software development process, more broadly.

Our study shows that, with the combined method, the CDNf perspective has boosted (and directed) the communicability inspection in ways (and directions) that we would not necessarily find with the MetacommT fill out alone. In particular, the choice of supporting progressive evaluation and its clashes with error proneness would not be formulated as such without CDNf input. In communicability terms, this might be nothing other than inconsistency in communication, rather than a design challenge, which is actually the case.

There are, however, at least two important questions to be discussed regarding our proposed method. One is the order in which the two methods are applied. Does it matter whether we start by inspecting usability issues or communicability issues? The other is the depth of the communicability inspection. Since there is a specific Semiotic Engineering method proposed to inspect the emission of metacommunication messages by the designers, why have we not used it in our study? What differences could we expect to find in the method itself and in the out coming results?

As of now we already have some preliminary answers to the above questions, although much remains to be investigated. Since 2012, we have been developing and improving a combined cognitive-semiotic evaluation method for software engineering tools. The method has already been applied for evaluating visual programming notations [10], application programming interface languages [1], as well as other modeling tools. [9] In some of our previous research, the order of semiotic and cognitive analyses was reversed, and in all of them we have used the full version of SIM, rather than the abbreviated MetacommT fill out used in the study reported here. Our abbreviated approach has emerged as a synthesis that allows us to gain efficiency in communicability evaluation for research purposes. We acknowledge that only experienced researchers who are familiar with Semiotic Engineering can use it productively. But, because there is so much more to uncover and discover in this field of research, we are not yet concerned with producing “off-the-self” solutions for software industry professionals to use right away.

**FINAL CONSIDERATIONS AND FUTURE WORK**

In this paper we have proposed a conceptual structure and method to support inspection-based evaluation of interaction with a particular modeling tool, namely IBM RSA. We have described the tool-notation-people triplet (conceptual structure) and the combined usability- and communicability-driven method with which we can obtain a rich characterization of the relations among people, tools and notations in typical modeling scenarios across software development processes. The triplet and the instantiation of the method were briefly illustrated with the use of IBM RSA in a UML activity model building scenario. We have shown the kinds of findings that can be obtained and discussed its value for modeling tool HCI design activities.

The study reported in this paper opens interesting paths for further investigations, some of which have already been mentioned in the discussion of our proposal (see the previous section). One such path is the investigation of how tools support model building compared to model use at later stages of the development process. What kinds of use of
and interaction with the model might other software developers have with a model somebody else has created?

Another interesting point to be investigated is the potential of model-drawing tools, compared to semantic modeling tools. If the user has the freedom to add new notations or expand the provided notation, we might see the emergence of socially-negotiated secondary notations that can fill up expressive gaps of the base notation supported by the tool. This might give us an insight into the social protocols that are sometimes used to complement (and compensate for the limitations of) technological protocols commonly used in collaborative tasks such as software development. [8]

Finally, we should not lose sight of why we are investigating the usability and communicability of modeling tools used in the software engineering process. Because models are in and of themselves a modeling tools used in the software engineering process, investigating the usability and communicability of supporting her research on this topic.

ACKNOWLEDGMENTS
Juliana Ferreira thanks CAPES for a PhD scholarship and Clarisse de Souza thanks CNPq and FAPERJ for partially supporting her research on this topic.

REFERENCES
15. OMG, Unified Modeling Language (UML), V2.4.1-http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF