"Through the glassy box": supporting appropriation in user communities.

Federico Cabitza and Carla Simone

Abstract Communities present considerable challenges for the design and application of supportive information technology (IT), especially in loosely-integrated and informal contexts, as it is often the case of Communities of Practice (CoP). An approach that actively supports user communities in the process of IT appropriation can help alleviate the impossibility of their members to rely on continuous professional support, and even enable complex forms of cooperative tailoring of their artifacts. The paper discusses the property of the accountability of IT applications as one of the basic enabling conditions for the appropriation of the technologies by their end-users, and for its most mature and sustainable form, that is End-User Development (EUD). We illustrate a framework, called Logic of Bricolage (LOB), proposed to both end-users and interested designers to describe (and make accountable) their EUD environments and systems, and facilitate both local appropriation and the sharing of experiences of IT adoption in CoPs.

1 Introduction

According to a oft-cited definition, "appropriation" can be seen as "the process by which people adopt and adapt technologies, fitting them into their working practices" (Dourish, 2003). Appropriation is a complex process whose success depends on the extent to what users are able to manage how "practices and technologies evolve around each other." (Dourish, 2001). As such, appropriation involves both social and technical concerns. From the social perspective (being appropriation but a form of practice), the notion of Communities of Practice (CoPs) (Wenger, 1998) characterizes the most favourable kind of work setting for a successful appropriation, as the negotiation and sharing of practices are part of the processes that constitute this kind of communities. From the

Universita degli Studi di Milano-Biccoca, e-mail: {cabitza,simone}@disco.unimib.it

technical perspective, the opacity of the technology is one of the main factors hindering a successful appropriation, especially when adaptation is concerned. Technological opacity can be generated by either the inherent complexity or rigidity of the technology; or by the way in which the technology has been actually constructed. In the first case, appropriation is necessarily bound to some workaround (Gasser, 1986): it is often the case that end users flank the complex, rigid (and often also imposed) technology by the so called *shadow tools* (Handel and Poltrock, 2011); these are simpler office applications, like spreadsheets and word processor templates, that are completely under the control of users and often are built in order to align with their situated practice and host the bunches of information that later users will transfer into the official applications of the information system at hand, as if this fictitious and often frail interoperability represented a post-hoc compliance with the organization policies (Bowers et al, 1995). Here we want to focus on the second case, which regards the development of cooperative applications in a community environment. Our paper then regards the opportunity to develop design strategies that, on one hand, could make opaque systems more transparent to their users; and, on the other hand, counteract the inevitability of the scenario where formal and informal tools coexist in the organizational domain, and hence the risk that they mutually undermine their function, especially in the long run.

1.1 Black and glossy boxes

In discussing the notion of context, Dourish (Dourish, 2003) made the point that when we focus on how users appropriate their software applications we should consider that the context is not something external to the applications, but rather something where the technologies are embedded, and where appropriation actually occurs. This would lead to reconsider the value, influentially advocated by Simon within the design sciences (Simon, 1996), of having technologies act as “black boxes” in the context of the user experience. In fact, this would rather shed light on the need to have tools that are more manifest to the involved actors, and become an observable part of the context where they make their decisions in regard to their adaptation, configuration and fit to their practices: like transparent boxes with respect to their inner functioning. Indeed, Dourish suggests three ways to facilitate appropriation: making users aware of the activities and the resources that are involved during their use of the system; making the system’s own structure and behaviors accessible to users; allowing them to define and negotiate the information structures, as well as their static organization and dynamic articulation. Since the first requirement would require reflection capabilities that hardly can be added to systems that are not purposely designed to host them, here we focus on the last two approaches, which advocate for more accountable systems, if not “self-accounting” ones. Thus, in this paper we characterize a purposely general framework that we called “Logic of Bricolage” (LOB) and first presented in (Cabitza et al, 2013). This framework is intended to support appropriation by users by giving them the “words” to make their systems more “accountable”. We use this term in the ethnomethodological sense of “observable and *reportable*”. Our point is that these two capabilities should not be decoupled: the appropriation of a car cannot be achieved only by making the car hood of transparent glass, as it is well known that showing low-level details of a computational system will not make it more accessible and comprehensible by end users, but probably just the opposite. The accountability we refer to is rather obtained by providing the concepts and words so that even users can denote “the system’s own structure and behaviors”, or at least cope with a representation of these that is suitable to be handled by lay users.

On the other hand, the LOB framework is also conceived for the developers, which in an EUD context can include also skilled end-users, as a conceptual model supporting the design of applications where structure and behaviors are clearly decoupled, and a clear separation of concerns is established between the process of “making the bricks” and the process of “assembling” them into walls and houses. This effort also should be paid for the better appropriation by end users, if not to make the system’s architecture cleaner and maintenance easier.

The paper first outlines the LOB framework, and presents the jargon it proposes to make EUD systems more accountable and transparent. Then we apply the framework to three existing EUD systems of applications, as an example of the post-hoc exercise in which existing systems are compared on a common conceptual ground. We then discuss the potential implications of adopting the LOB, or any equivalent, framework in EUD practice to foster negotiation (that is mainly a discursive practice) and appropriation in communities of users.

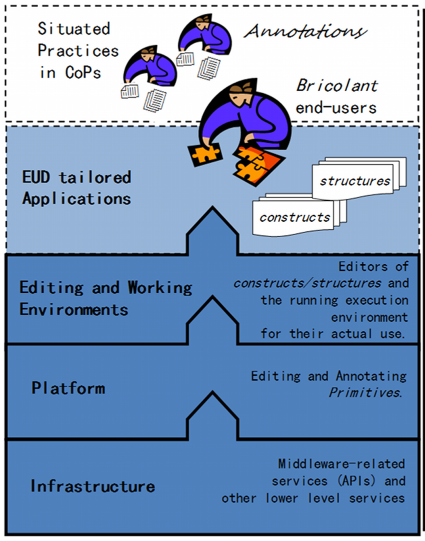
2 A primer on the Logic of Bricolage

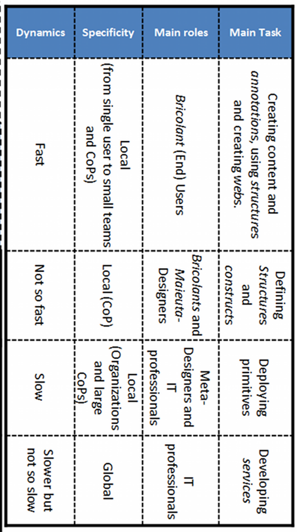
In (Halverson et al, 2008) Halverson has aptly proposed to evaluate the utility of any conceptual proposal, or theory, in the design of cooperative applications considering its potential on different planes of *power* (or afforded capabilities). After this contribution, we propose the LOB framework as an approach that can: i) Facilitate practitioners in making sense of and describing their and others’ systems. This regards to the *descriptive power* of frameworks and it applies to both communities of end-users, who make a common sense of what it is supplied to their community, and to communities of designers, who are supposed to present and make their solutions understandable within their reference community of IT practitioners. Similarly, ii) help designers talk about their solutions by providing them with a common vocabulary, i.e., a very concise lexicon whose available terms cover few but essential aspects of recurring EUD conceptual structures and underlying models and are defined with some degree of unambiguous formalization. This regards the so called *rhetorical power* of LOB, for which it is aimed at facilitating the sharing of lessons learned, best practices and effective solutions, also on a narrative level (Orr, 1996). Lastly, iii) both inform and guide the design of EUD proposals that could meet the challenging requirement to let the members of an end-users community develop and maintain practices of technology local tailoring and adaptation to their emerging and ever-changing needs. This regards the *applicative power* and is by far the most difficult "power" to sustain although it is the most precious one in terms of impact on real life practice.

As the terminological dimension is important for for the intended powers of the LOB framework, in the rest of the section we will characterize the terms and concepts that make it useful in accounting for technological EUD applications. The expression *Logic of Bricolage* itself is chosen after (Lanzara, 1999) to denote the articulated environment, or application context, where end-users are called to *co-define* and *use* tools, to, respectively, build computational structures and their behaviors (*editing environments*), and to have those behaviors be executed at run time (*execution environment*). It therefore refers to a context endowed with some order and logic, but where the main valuable activity is an "orderly patchworking" and assemblage of pieces. Since the framework has been already presented in (Cabitza et al, 2013) and its formalization through a generative grammar has been proposed in (Cabitza and Simone, 2014), in what follows, we will just recall the main concepts and elements.

The elements characterizing the LOB framework can be arranged into a conceptual architecture, that we depict in Figure 1 where LOB terms are in italics and for each layer, its name and what it offers to the higher layers are specified. From the topmost layer of this architecture, we see as end users are enabled to create and use community-specific *applications* by interacting with the *editing and working environments*: to this aim, these environments expose apt building blocks (called *constructs*) through *specific editors* that are supported by the underlying EUD *platform* in terms of *primitives.* These latter are domain-independent functionalities that are expressed in terms of lower level Application Programming Interfaces (API). The platform in its turn is enabled by a regular infrastructure (i.e., an application server and operating system).

According to the bottom-up approach advocated within the LOB framework, *constructs* have to be identified during the inception phase of the framework for a specific end-user community, as a result of the interaction between its members and the IT professionals: these are also in charge of construction of the above mentioned API and primitives. After this point, end-users should be able to construct, tailor and appropriate applications through incremental and actually never-ending task-artifact cycles (Carroll et al, 1991) in which members of this community agree over time upon what constraints and functionalities must be enacted by their supporting technology.


Situated Practices in CoPs
Annotations
Bricolant end-users
EUD tailored Applications
structures
constructs
Editing and Working Environments
Editors of *constructs/structures* and the running execution environment for their actual use.
Platform
Editing and Annotating *Primitives*.
Infrastructure
Middleware-related services (APIs) and other lower level services



Dynamics	Specificity	Main roles	Main Task
Fast	Local (from single user to small teams and CoPs)	*Bricolant* (End) Users	Creating content and *annotations*, using *structures* and creating *webs*.
Not so fast	Local (CoP)	*Bricolants* and *Maieuta-*Designers	Defining *Structures* and *constructs*
Slow	Local (Organizations and large CoPs)	Meta-Designers and IT professionals	Deploying primitives
Slower but not so slow	Global	IT professionals	Developing *services*

Fig. 1: A conceptual architecture for EUD environments supporting bricolage.

Constructs are distinguished between *Operand Constructs* and *Operator Constructs*: operands are the most atomic data structures that make sense in a specific domain; operators are all the micro-functions that are deemed necessary to be performed over the operands in the application domain; operators can be either *functional* or *actional*, to modify the value of the operands or to produce some effect in the computational environment (like. triggering a communication), respectively; in particular, (functional) Operator Constructs can be applied to operands to allow for the recursive construction of more complex operands from simpler ones.

End-users can arrange and compose together in a bottom-up fashion both kinds of *constructs* through two classes of editors, that define respectively, the rules of constructs composition to build the working spaces and artifacts, and their computational behaviors: these compositions are called *structures*, more precisely *Layout structures* and *Control structures*. In simple terms, *Control structures* specify the behaviors of *Layout structures*, i.e., how the artifact acts on the content inscribed therein, e.g., in response to events generated at interface level, and how this level interacts with users during the habitual use of the application.

Layout structures shape the "work spaces" that are recognized by end-users as constituted by the physically inscribed and computationally augmented artifacts where and by which they carry out their work. For example, in the domain of computer-aided design and collaborative drawing/editing, a Layout Structure is the working space where users arrange the docking bars of their preferred commands, symbol stencils and predefined configurations of elements that must be set up before the actual work begins. In document-based information systems, Layout Structures are the document templates of forms and charts that are used to both accumulate content and coordinate activities ; such structures are endowed of both physical properties and symbolic properties, for instance, input controls (i.e., data fields), boilerplate text, iconic elements and any visual affordances conveyed through the graphical interface, which in LOB are all instances of Operand constructs. Layout structures result from the topological arrangement of Operand constructs. Layout structures can be aggregated in *Web Structures*, that are recursively defined as interconnected sets of Layout Structures.

Also Control Structures are recursively defined in terms of simpler rewriting rules, that is sort of "conditioned actions", which are expressed in terms of specific Operator Constructs. Conditions are expressed as functional constructs applied to the current state of computation that encompasses all application data. Control Structures can be of arbitrary complexity, from simple rules to composition of instructions in virtue of a special kind of operators (called *Connectors*). Control Structures are interpreted by the execution environment: by interpreting the Operator Constructs constituting them as more or less complex articulation of primitives (see Figure 1).

The last feature offered by the LOB framework is the possibility to build *annotations*, that is any user-defined content that is created to be anchored to any another content. Notably, in LOB annotations can be nested, that is users should be able to *annotate annotations*, so as to allow for nested threads of comments and tags, as we described in [anonymized]. Annotations are conceived as pieces of a collaborative and never-really-finished bricolage, which hosts informal communication and handover between practitioners, their silent and ungoverned work of meaning reconciliation, and the sedimentations of habits and customs in effective (yet still unsupported computationally) conventions of cooperative work (Cabitza et al, 2009a). Also Control Structures can be annotated to support their collaborative construction (Cabitza and Simone, 2012, p. 232).

3 LOB as a general framework to describe existing applications

In order to support our claim that the LOB can play as a general framework where to express environments supporting various kinds of EUD applications and make their structure more accountable, this section provides three examples of how the LOB terms and concepts can be used to describe different concrete environments: to make the above mentioned claim general enough, we consider three heterogeneous environments: first a document-based collaborative system designed to be highly tailorable according to the local work settings; then an environment that allows for the user defined integration of devices and software components supporting groups of cooperating actors; and lastly, an example of a multi-layered and flexible mashup composition environment that allows for the integration of data sources and functional components to produce enriched and personalized results. These three situations, that are bound together by the collaborative nature of the work that members of specific communities carry on, cover a significant amount of concrete cases where users express the requirement to be in full control of the development of their computational tools and make it a collective, incremental and often bottom up, spontaneous process.

3.1 WOAD: constructing Webs of Active Documents

On the basis of field studies conducted in different work settings, especially in the healthcare domain, a document-based collaborative system, called Web of Active Documents (WOAD) has been proposed in (Cabitza and Simone, 2010). The core concepts of WOAD can be summarized as follows in terms of: i) the information structure is composed of hyperlinked active documents that can be annotated in every parts and sections and be associated with any other document, comment and computational behavior; ii) there is no rational and unified data model: users define their forms in a bottom up manner and, in so doing, the platform instantiates the underlying flat data structures that are necessary to store the content these forms will contain and to retrieve the full history of the process of filling in them; iii) the presentation layer is in full control of end users, who are called to both generate their own templates and specify how their appearance should change later in use under particular conditions by means mechanisms that are expressed in terms of if-then rules. Users can define local rules that act on the documents' content and, as hinted above, change how documents look like (i.e., their physical affordances), to make themselves aware of pertinent conditions according to some cooperative convention or business rule like, e.g., the need to revise the content of a form, or to consider it provisional, or to carefully consider some contextual condition . The LOB conceptual architecture offers a framework that incorporates the various WOAD's components in a coherent picture: in the following we associate the concrete items constituting WOAD with items of the LOB framework.

First, we specialize the constructs: remember that these are application domain dependent and therefore they have to be defined by the users in cooperation with designers and, when necessary, with IT professionals according to the needs of the specific application domain.

The **Operand constructs** in WOAD are called datoms (document atom): these are but any writable area with a unique name and a type (e.g., Integer, String). A datom can recursively be a composition of one or more datoms: e.g., the 'first name' datom (a string) and the 'family name' one (also a string) can be combined into a 'person name' datom that encompasses both. The

Operator constructs are a selection of atomic functions: for example, as reported in (Cabitza et al, 2009a) doctors from a medical setting required, besides the standard arithmetic and boolean operations, also a construct to perform the *average*, and another one checking the occurrence of a value in a given set (the *is-in* construct). A list of Actional Operator Constructs conceived to be applied to the Layout Structures or on their components has been derived from a series of field studies and it encompasses basic operations like *save*, *retrieve* and *store* ; one of these constructs, namely *annotate*, can associate a didget (i.e., a document widget resulting from the instantiation of a datom) with an annotation. More complex operator constructs can be recursively defined by composing more elemental ones. **Web structures** are a graph of hyperlinked *templates*, i.e., WOAD **Layout Structures**; these latter are a set of didgets,: a didget is a topological object, i.e., an Operand construct called datom (see above) that is put in some place, i.e., is coupled to a set of coordinates (that in WOAD are represented as Cartesian pairs with respect to the origin of the template). It is worthy of note the fact that the two datoms mentioned above (first name, family name) can be used to create a WOAD template (i.e., a Layout Structure), as well as a third datom, i.e., another Operand construct: in the former case the two-datom set is to be used in the execution environment in documentts that are instances of the template encompassing it; in the latter case the set is intended to be used atomicaly as component itself of other templates (i.e., as a topological object) in the editing environment. In WOAD the **Control structures** are called *mechanisms*, i.e., if-then rules whose if-part is a Boolean expression that is recursively defined using the predefined datom names as variables, and the operators identified above, all together with the (obvious) constants of the basic types. The then-part is a sequence of actions that has to be performed on the template or on its inner components. Mechanisms are connected by the (implicit) OR connector. In WOAD, the **Primitives** that allow for the definition of both Layout and Control Structures are the following ones: *aggregate*, to build complex operands from simpler ones; *compose*, to build complex operators in terms of functional composition; *localize*, to associate a didget with a Cartesian coordinate with respect to the origin of a given template; *list*, to build sequences of if-then rules; Moreover, the *annotate* primitive associates a text with an operand.construct.

As mentioned in the previous section, the Primitives are offered through an editing environment where Constructs and Structures can be defined. In WOAD this environment is constituted by two visual editors: one for the construction of mechanisms , and one for the construction of datoms and, by arranging these latter topologically in terms of didgets, templates.

3.2 CASMAS: creating hybrid communities for cooperation.

Suppose that a set of applications and devices have to be integrated to support a set of actors that cooperate by means of them. According to the Community-Aware Multi Agent Systems (CASMAS) framework (Locatelli and Simone, 2012), both actors and their tools all are represented as *entities* and integration can be seen alike to becoming members of the same *community*: as such, they coordinate their behaviors through a shared information space that contains coordinative information, as well as the *behaviors* that are dynamically assigned to each entity to make it an active member of the community: in CASMAS communication is asynchronous but it is not message based. Instead, when an entity *posts* a request into the space, other entities will *react* to this request according to their current behaviors.

The CASMAS framework encompasses a language to specify entities and their behaviors. This language takes the declarative form of facts and rules (if-then patterns), which offers the possibility to express behaviors in a modular way, without the need to define complex and exhaustive control structures (Myers et al, 2004). The rules constituting an entity’s behavior express *what* the entity is expected to do *when* some conditions are satisfied: these conditions are matched against the facts contained in the community’s space and in the entity’s local memory; the action(s) that the entity should perform updates either the community space or the memory of the entity itself.

The integration of a software application/device is realized by inserting a fact in the memory of the entity representing it and by defining the behavior of this entity. The fact contains attribute-value pairs that specify the information the application/device makes available for sake of coordination with the other entities of the same community; the entity’s behavior expresses conditions (among others) on the concrete application/device attributes (when) and invokes some of the functions the application/device exposes to the community (what): actually, the entity is a sort of wrapper that mediates between the concrete application/device and the integration environment (community).

As done for the WOAD framework, we associate each CASMAS feature with each item of the LOB framework.

In CASMAS, the **Operand constructs** are the facts that are contained in and exchanged across community space(s): CASMAS facts are expressed according to the syntax of the underlying rule-based language (currently, JBoss Drools[1].). The **Operator constructs** are the basic functions and predicates that are exposed by the underlying rule-based programming language;in particular the actional operator constructs (actions in CASMAS) support the asynchronous communication between entities as well as the storing and retrieval of information among the spaces and the local memories. In case of applications/devices’ memories, the store/retrieve actions respectively put and

[1] `http://www.jboss.org/drools/`

get information to/from the data structures therein managed. The *spaces* that are implicitly connected through the entities that are members of more than one community are the LOB **Web structures**; each space is a **Layout structure** that contains the community's facts and the entities' ones; differently from WOAD, facts (i.e., operand constructs) are not geometrically localized within spaces, as CASMAS does not specify the coordinates of its topological objects (i.e. the facts within the spaces). The if-then rules connected through the OR connector and grouped according to the entities membership to the community are the CASMAS **Control structures**: their if-parts encompass sets of Operand and Operator constructs, as in WOAD; similarly, the then-part can either encompass the above mentioned *put* and *get* actional operator constructs, whenever the behavior regards applications/devices entities; or a *post* action, in the other cases. The CASMAS framework defines the same primitives seen for WOAD (except for the *localize* and *annotate* primitives), but it also encompasses the *put* and *get* primitives: the role of these primitives is to interact with the wrappers developed for each devices/application to be integrated and they are called in the actions having the same name.

3.3 DashMash: flexible configuration of EUD mashups

Recently, an increasing number of environments where users can combine information flows from different data sources, the so called mashups, has been proposed, also for commercial use (e.g., Yahoo Pipes). For sake of exemplification, we apply our exercise of LOB instantiation to the the DashMash framework (Cappiello et al, 2011)[2], which we take as representative of a wide class of applications that allow for the collaborative user-driven aggregation of heterogeneous content. Indeed, *DashMash* is a general-purpose EUD environment that adopts an approach in which the design-time and the use-time are strictly intertwined: end-users can autonomously define their own mashups and execute these latter "on the fly", to progressively check the result of their editing activities. Like most of the traditional approaches for the creation of mashups, also the DashMash approach is dataflow-oriented, i.e., end-users can only aggregate, filter and display data in the most meaningful way, e.g., a pie chart, a table or a map. On the other hand, the DashMash approach gives also the possibility to provide end-users with an environment that can be customized so as to meet their domain-specific requirements; essentially, this can be done in two ways: (i) through the development of domain-specific components that allow to interact with the functionalities provided by any kind of (local or remote) service; and (ii) providing end-users with the access to data coming from private and domain-specific data sources, in addition to publicly accessible ones. Nevertheless, the approach used in DashMash

[2] This task is less detailed than in the other two cases as the mentioned paper does not give all the necessary details.

provides end-users with an abstraction that makes them able to use the various mashup components (e.g., data sources, filters and data viewers) that are automatically composted on the basis of a pre-defined set of compatibility constraints, relieving end-users from the need to know any technical detail about the used components.

As for CASMAS, the DashMash **Control structures** are grouped to form the behavior of each *component*. These constructs allow for typical publish and subscribe patterns, like "if a new fact occurs, then publish an event" and "if a subscribed event occurs, then perform some operations". New facts or operations pertain to single components only. For example, if the component is the *Composition Handler*, then the new fact is any change in a component; the components influenced by this kind of event (i.e., the subscribers) activate the corresponding operations: for example, if the change is about a Filter Component then all Data Components using this filter activate internal operations to send the data to the new Filter Component and at the end this later notifies that new-data are ready: this event is consumed by the Viewer Component subscribing this event for the specific data. In this view, the **Operand constructs** are the *data* and the *events*, while the main **Operators constructs** regard the publication of an event, and the subscription for a specific kind of event. In DashMash, the Web structure is the set of workspaces; each workspace is a **Layout structure** that is composed by two inner Layout structures: one contains the output of all the Viewer Components for what concerns the data (i.e., at the use level according to (Ardito et al, 2012)); the second contains a standard description of the workspace *state* in terms of Components such as: Data Sources, Filters and Viewers (i.e., at the design level).

More traditional mashups that, differently from DashMash, are uniquely based on data flows can be described as graphs whose nodes are input-output transformations, and whose arcs express the kind of connection that hold between two nodes. In LOB terms, a mashup belonging to this class can be seen as a set of rewriting rules that transform inputs into outputs, where arcs are as connectors that express the appropriate structure of the data flow (e.g., either alternative or parallel flows).

Table 1 highlights how the LOB approach applies to the the three frameworks characterized above. These three instantiations support our claim that the LOB architecture is at the same time general enough to formally describe different types of EUD application classes (e.g., information mashup, document-based systems, integration of applications), and yet detailed enough to define a concrete platform to apply recurring design patterns for EUD systems to be deployed in different application domains.

Table 1: Synoptic table of LOB concepts applied to the frameworks analyzed.

Framework	Primitives	Constructs	Structures	Annotations
WOAD	*aggregate, annotate, compose, list* and *localize*	*annotate, attach, average, cache, copy, correct, count, create, datom, delete, is-in, officialize, open/read, print, protect, retrieve, save, select, store, transmit* and *write*	*mechanisms* and *templates*	Yes
CASMAS	*aggregate, compose, get, list and put*	*get, post, put*, rule patterns and facts	*space* and *behaviors*	No
DashMash	data and events	*publish, subscribe*, and components, data sets of and events	*workspaces* and *workspaces*	No

4 Discussion

In our aims, the previous section would show that, should the LOB framework fall short of demonstrating applicative power to the test of life, it can at least foster a scholarly debate, in virtue of its descriptive and rhetorical powers, on the need of having more frameworks with similar scope and aims. This should be true especially in the hybrid field where CoP-oriented and EUD-related concerns meet, and towards the dissemination of these concerns in multiple venues, research initiatives and digitization projects. Nowadays this need should be particularly felt especially by those researchers that espous the main tenet of the End User Development field (i.e., the idea that computational artifacts should be increasingly developed by end-users themselves), as to date this idea has not yet gained in popularity in IT production, let alone in regard to communities support. A seminal analysis of the reason for this gap between research and practice asserts that the approaches up to now “have not been developed to cover end users’ entire scope of work” (Syrjaenen and Kuutti, 2011): this work is primarily social and deeply grounded in communities of practice.

As stressed above, appropriation plays an important role in the perpetual evolution of these communities, as it regards patterns of technology *adoption* and *adaptation* that can only be learnt through situated practice and social participation, as well as through “the transformation of practice at a deeper level” than the mere customization (Dourish, 2003). Moreover, communities appropriate the technologies that mediate interactions among their members in complex and partly unanticipated ways (Pipek and Kahler, 2006); this is because this process is intertwined with a great deal of invisible work, tacit knowledge, conventions, habits and mutual expectations, all essential elements in the constitution of communities of practice. As appropriation concretely relates also to specific ways to configure, adapt and tailor technologies, we share the wonder by Bodker (Bodker, 2006), who observed how end-user tailoring is seldom taken in serious consideration when speaking of “design for communities”. Here we are referring to tailoring not in terms of the individual adaptation of technology for personal use but rather to the “adaptation and further development [of computational technologies] through interaction and cooperation among people”, which calls for specific methods and environments that enable end-users to create and maintain their own tools.

Thus, we observe a paradoxical phenomenon: on the one hand, designing technologies for CoPs is seldom articulated in terms of enabling their members to autonomously build and shape their own tools, that is the main concerns of EUD research (Pipek et al, 2009). On the other hand, EUD research seldom takes communities of users as a first class concept to fully account for the fact that end-users are most of the times members of complex social ensembles. The LOB framework aims to contribute to bridge this apparent gap by focusing on the need to improve both the accountability of the technologies, as well as of the methods and environments where these are built and appropriated.

To this regard, some differences and complementarities of LOB with respect to existing frameworks can be found and discussed. LOB shares some strong affinities with the concept of meta-design proposed in (Fischer and Scharff, 2000), and some affinities with the approach based on component design (Won et al, 2006), and with the approach described in (Costabile et al, 2007), which all acknowledge the substantial continuity between design and usage of software applications. Meta-design, in particular, is one of the most complete approaches to EUD, but it seems to formulate general principles that do not really consider the peculiarities of designing systems that must be appropriated in communities of practice. For instance meta-design does not consider the importance of annotations, which conversely LOB takes as the first kind of appropriation “at the surface” of the applications’ interface. It does not consider the need of a formal language where items are defined in terms of others, that conversely LOB takes seriously as a way to mirror in computational structures how composite EUD applications can be composed of smaller building blocks. And lastly, meta-design to date does not articulate the two roles of EUD initiatives, i.e., users and designers with the fine-grained details of the LOB vocabulary that distinguishes between more passive end users (bricoleur), active end users (bricolant), more community-oriented designers (maieuta-designers) and designers more concerned with the technological platform (see Figure 1 and (Cabitza et al, 2014) for more details on this comparison).

On the other hand, the LOB framework shares with Component design the focus on modularity, but differs from it with regard to the limits to end-user tailorability in the introduction of opaque components. The approach based on Software Shaping Workshops (SSWs) described in (Costabile et al, 2007) , unlike our proposal, is more aimed at the definition of an organization structure and of a methodology for EUD design, rather than at the definition of a conceptual framework and architecture supporting the description and shaping of each possible EUD environment; even more importantly, such an approach is strongly oriented at the shaping of the interaction of users with their tools (which are however constructed by IT professionals), rather than at informing the users' activity of autonomously defining their tools themselves. This is important especially in a community-oriented perspective since, although embedded in structured organizational settings, communities are sort of "autonomous" bodies (with respect to top-down ordering initiatives) within the scope of their constituting practices and artifact use.

5 Concluding remarks

In this paper we have presented the LOB framework, as a design-oriented tool to distinguish and separate concerns in the conception and development of EUD-enabling platforms, associate these concerns with specific layers of a common reference architecture, and call objects pertaining to each layer with specific and evocative names, by following the precept to "keep it simple, but not simpler".

The reason why we presented such a framework is that we are convinced that in EUD "the best is yet to come". This claim is not to discard what has been done so far in this research field; on the contrary it is an invitation to recognize that the solutions that have been brought forth to allow end-users to create and maintain their computational tools autonomously have now reached a maturity level that *requires* a sort of backward reflection, as well as an effort to generalize local solutions and intuitions into general insights and concepts that could enable future reuse and discussion, especially for their application into real settings. We also believe that the key factor for this to happen is to "scale up" the experiences collected in the last ten years or so of research in EUD. In this regard, we see *appropriation* as the learning process by which each member of an end-user community understands what a technology can do for herself, and how it enables, constrains and shapes the community's practices, often well beyond the intentions of the technology designers. In this process, we have argued in this paper that the component-level accountability of the technology (in the ethnomethodological sense) is a basic requirement to pursue.

Unnecessary implementation details can (should!) be still made opaque to end-users, provided that they can *have the illusion* of being looking at the real "nuts and bolts" of their cars, and of being enabled to tweak and tune them up for the better functioning of their tools. This is what all EUD platforms ultimately aim to: to empower passive and "incompetent" users into "bricoleurs" that move and assemble what surround them in new forms of support. The creative potential of technological bricolage in organizations could also be seen as a thinking-out-of-the-box solution to cope with the current conundrum of having ever increasingly complicated information systems dealing with increasingly significant and complex portions of the socio-technical contexts in which they operate, thus increasing the risk of potentially serious and completely unintended consequences (as the risk of misalignment between the official information system and its shadow tools is just an example). Scattering functions and information nuggets in highly connected networks of processes and working spaces, respectively, with denser clusters that mirror responsibility and ownership boundaries of specific communities of end-users could be a direction to investigate further, instead of looking for forms of more or less disguised centralized control that still characterizes most of the current information systems.

This is the main direction we also aim to take in the further development of the Logic Of Bricolage that we presented in this paper as a contribution towards a feasibly ordered bricolage of partly persistent and partly transient structures that scaffold knowledge work and support cooperative work in real communities and organizations.

References

Ardito C, Buono P, Costabile MF, Lanzilotti R, Piccinno A (2012) End users as co-designers of their own tools and products. JVLC 23(2):78—90.

Bowers J, Button G, Sharrock W (1995) Workflow from within and without: Technology and cooperative work on the print industry shopfloor. In: ECSCW'95:, Kluwer Academic Publishers, pp 51–66

Bodker S (2006) When second wave HCI meets third wave challenges. ACM Press, pp 1–8

Cabitza F, Simone C (2010) WOAD: a framework to enable the end-user development of coordination oriented functionalities. JOEUC 22(2):1–20, IGI Global

Cabitza F, Simone C (2012) Affording mechanisms: An integrated view of coordination and knowledge management. CSCW 21(2):227—260

Cabitza F, Simone C (2014) Building socially embedded technologies: Implications on design. In Designing Socially Embedded Technologies, Springer, forthcoming

Cabitza F, Simone C, Sarini M (2009a) Leveraging coordinative conventions to promote collaboration awareness. CSCW 18(4):301–330

Cabitza F, Simone C, Gesso I (2013) Back to the future of EUD: the logic of bricolage for the paving of EUD roadmaps. In IS-EUD 2013, Springer, LNCS, vol 7897, pp 254–259

Cabitza F, Fogli D, Piccinno A (2014) "Each to his own": Distinguishing activities, roles and artifacts in EUD practices. In: Empowering society through digital innovations, LNISO, Springer

Cappiello C, Daniel F, Matera M, Picozzi M, Weiss M (2011) Enabling end user development through mashups: Requirements, abstractions and innovation toolkits. LNCS, vol 6654, Springer, pp 9–24

Carroll JM, Kellogg WA, Rosson MB (1991) The task-artifact cycle. In Designing Interaction: Psychology at the Human-Computer Interface, Cambridge University Press, pp 74–102

Costabile MF, Fogli D, Mussio P, Piccinno A (2007) Visual interactive systems for end-user development: A model-based design methodology. Transactions on Systems, Man and Cybernetics, 37(6) IEEE

Dourish P (2001) Where the Action Is: The Foundations of Embodied Interaction. MIT Press, Cambridge, USA

Dourish P (2003) The appropriation of interactive technologies: Some lessons from placeless documents. CSCW 12:465–490

Fischer G, Scharff E (2000) Meta-design: Design for designers. In: DIS'00 ACM, New York, NY, USA, pp 396–405

Gasser L (1986) The integration of computing and routine work. ACM TOIS 4(3):205–225

Halverson C, Ackerman M, Erickson T, Kellogg WA (eds) (2008) Resources, Co-Evolution and Artifacts: Theory in CSCW, Springer

Handel MJ, Poltrock S (2011) Working around official applications: Experiences from a large engineering project. In CSCW'11: ACM, pp 309–312

Lanzara G (1999) Between transient constructs and persistent structures: designing systems in action. Journal of Strategic Information Systems 8(331-349)

Locatelli MP, Simone C (2012) End-users' integration of applications and devices: A cooperation based approach. In From Research to Practice in the Design of Cooperative Systems: Results and Open Challenges, Springer, pp 167—181

Myers BA, Pane JF, Ko A (2004) Natural programming languages and environments. Communication of the ACM 47(9):47–52

Orr J (1996) Talking about Machines: An Ethnography of a Modern Job. Cornell University Press

Pipek V, Kahler H (2006) Supporting collaborative tailoring. In End User Development, vol 9, Springer

Pipek V, Rosson MB, Stevens G, Wulf V (2009) Supporting the appropriation of ICT: end-user development in civil societies. In Learning in Communities, Springer , pp 25–27

Simon HA (1996) The sciences of the artificial, 3rd edn. MIT Press, Cambridge, Mass

Syrjaenen AL, Kuutti K (2011) From technology to domain. ACM Press, pp 244–251

Wenger E (1998) Communities of Practice: Learning, Meaning, and Identity. Cambridge University Press, Cambridge, U.K.

Won M, Stiemerling O, Wulf V (2006) Component-based approaches to tailorable systems. In: End User Development, vol 9, Springer, p 115–141