On Policies for Bartering Access to Resources[⋆]

Lorenzo Bettini[1,†], Rosario Pugliese[1,†] and Francesco Tiezzi[1,*,†]

[1]*Dipartimento di Statistica, Informatica, Applicazioni, Università degli Studi di Firenze, Firenze, Italy*

Abstract

Resource access management in modern ICT systems usually authorizes access to resources based on factors like identities, roles, and possibly attributes of the requester, the requested resource, and the environment. However, traditional approaches to access control do not fit well with collaborative scenarios where, typically, users can define their own policies allowing access to their resources provided that some conditions on the accessibility of other resources are met. In this paper, we introduce Bart, a language to specify attribute-based access policies for bartering resource access among the parties of a distributed system. Using Bart, each party can independently define policies to control access to their resources, including specific conditions that may require agreement with other parties to access additional resources. Subsequently, based on the policies currently in force within the system, the evaluation process – triggered on demand to determine whether an access request can be granted – automatically manages the bartering of resource access among policies.

Keywords

Attribute-based Access Control, Policy Languages, Collaborative Scenarios

1. Introduction

Resource security is a cornerstone of modern Information and Communications Technology (ICT) systems. Ensuring that an organization's resources are accessible only to authorized users is essential to prevent unauthorized or malicious access. At the same time, in today's competitive landscape, organizations often need to collaborate to drive innovation and maintain their edge. In such scenarios, resource sharing – encompassing data, services, digital assets, and computational resources – becomes fundamental. The challenge lies in facilitating this exchange in a secure, controlled, and automated manner that fosters trust and efficiency.

Access control systems serve as the first line of defense for securing ICT systems. These systems determine whether a *subject*'s *request* for access to a *resource*, such as reading data or logging into a server, should be permitted or denied based on predefined conditions. Over time, various access control approaches have been proposed. Traditional methods rely on the identity of subjects, either directly – e.g., Access Control Matrix [1] – or indirectly via predefined roles or groups – e.g., Role-Based Access Control (RBAC) [2]. A more recent approach, Attribute-Based Access Control (ABAC)[3], leverages the notion of *attributes* for representing security-relevant information about the system, subjects, resources, actions, and the environment. ABAC enables the creation of fine-grained, flexible, and context-aware access control rules that are expressive enough to uniformly represent all the other approaches [4]. Typically, ABAC rules are structured into *policies*, leading to the term Policy-Based Access Control (PBAC) [5], which is sometimes used interchangeably with ABAC.

Despite their diversity and prevalence, traditional access control approaches and their associated policy languages are not well-suited for collaborative scenarios. In such contexts, users often need to define their own policies, granting access to their resources only if specific conditions on the accessibility of other resources – potentially generating additional requests and triggering the evaluation of other policies – are met. Traditional policy languages, however, are limited to expressing conditions based

Joint National Conference on Cybersecurity (ITASEC & SERICS 2025), February 03-8, 2025, Bologna, IT

[⋆]This work was partially supported by project SERICS (PE00000014) under the MUR NRRP funded by the EU - NGEU.

[*]Corresponding author.

[†]These authors contributed equally.

lorenzo.bettini@unifi.it (L. Bettini); rosario.pugliese@unifi.it (R. Pugliese); francesco.tiezzi@unifi.it (F. Tiezzi)

0000-0002-4481-8096 (L. Bettini); 0000-0002-1419-1405 (R. Pugliese); 0000-0003-4740-7521 (F. Tiezzi)

on factors such as the identities, roles, and attributes of the requester, the requested resource, and the environment. They lack the capability to define *side conditions* that depend on the accessibility of other resources. In collaborative scenarios, access to resources is typically negotiated in advance outside the access control system. The resulting policies are then implemented manually and enforced through the authorization system, leading to inefficiencies and reduced flexibility.

Consider, for instance, a social network where users can define access policies for their resources, such as photos. Let us examine the policies of two users, Alice and Bob: Alice permits access to her photos only to those who, in return, are willing to share their photos with her; in contrast, Bob adopts a more permissive policy, allowing anyone to view his photos without restrictions. What Alice is proposing is an access policy aimed at a sort of mutual benefit. Considering the policies described by the two, it is clear that both Alice and Bob should be allowed access to each other's photos. However, it is impossible to render Alice's policy in traditional policy languages, despite the remarkable expressiveness of such languages as, e.g., the OASIS standard eXtensible Access Control Markup Language (XACML) [6]. Instead, a 'third party' should first manage the original policies of Alice and Bob separately from the access control system for evaluating the side conditions and then express the policies deprived of the side conditions as, e.g., XACML policies. This task, however, is complex as several users and policies could be involved, error-prone as it is done manually, and requires the existence of a trusted third party. Moreover, any changes to the original users' policies would lead to the need to reconsider and, in the worst case, rewrite the resulting access control policies to be enforced.

To overcome these limitations, in this paper, we introduce Bart, a language to specify attribute-based access policies for bartering resource access among the parties of a distributed system and set them in an automated and system-managed way. Using Bart, each party can independently define policies to control access to his/her resources. These policies may include specific conditions that require agreement with other parties to access additional resources in exchange. Subsequently, based on the policies currently in force within the system, the evaluation process – triggered on demand to determine whether an access request can be granted – automatically manages the bartering of resource access among policies. This eliminates the need for pre-arranged agreements outside the access control system. Access requests are evaluated dynamically based on the current policies in force without altering the policies themselves. This ensures that resource owners retain full control over their policies and can modify them – or their associated conditions – at any time as the system evolves, offering both flexibility and adaptability.

We will present the language and its evaluation process informally through a few examples. Intuitively, a Bart specification is a sequence of policies, one for each party of the considered collaborative system. Each policy is a list of attributes characterizing the party that owns the policy and a list of rules. Each rule governs access to a specific resource by (*i*) enforcing checks on the values of the attributes required from access requests or the working environment and (*ii*) possibly specifying reciprocal access requirements, detailing which parties must provide access to which resources in exchange for granting access to the requested resource.

The rest of the paper is organized as follows. Sec. 2 illustrates the Bart policies of a simple collaborative scenario and shows how the request evaluation process uses them, thus providing a presentation of the distinctive features of Bart. Sec. 3 describes other scenarios from the same case study; they enrich the initial scenario to illustrate specific technicalities of the Bart language and how the approach applies to increasingly complex situations. Sec. 4 presents an overview of the most strictly related work, while Sec. 5 concludes and touches upon directions for future work.

2. The Bart language

In this section, we introduce the Bart policy language by resorting to a delivery case study, used throughout the paper as a running example. The formal definition of Bart is out of the scope of this paper (the formal syntax and semantics of Bart are reported in [7]).

Language features. A Bart specification is defined by a sequence of policies, called a *policy system*.

A *policy* (party : *Attributes*, rules : *Rules*) defines the access rules for the resources controlled by a single party of the policy system. A policy is defined as a pair of labeled fields: the first field (party) specifies a non-empty list of attributes that characterize the party and permit its identification, while the second field (rules) specifies a list of access rules.

An *attribute* is a pair of the form (*name* : *value*), where *name* is a string that refers to the literal *value* associated with a characteristic of the involved parties, resources, or any other relevant entity. We commonly use values such as booleans, numbers, strings, dates, and sets of these values. We also assume no *name* occurs multiple times as an attribute name in any attribute list.

A *rule* (resource : *Attributes*, condition : *Expression*, exchange : *Exchanges*) is defined as a triple of labeled fields: the first field (resource) specifies a non-empty list of attributes characterizing a resource controlled by the rule's owner party; the second field (condition, which, if omitted, always evaluates to true) specifies an expression indicating the condition that must be satisfied for granting the access to the resource; the third field (exchange), if present, specifies which accesses to which resources for which parties the rule's owner party requires in exchange for granting access to the resource. The syntax of *expressions* is deliberately left underspecified; we just assume that expressions are built from attribute names and values by using standard logical, relational, and arithmetic operators. Notably, attribute names in expressions can refer to attributes specified in the request under evaluation, the requester's field party, or the context (the notion of context will be introduced later).

An *exchange* is a combination using the and and or boolean operators of triples of labeled fields (to : *To*, resource : *Attributes*, from : *From*). The first field (to) specifies to which parties the resource access in exchange has to be granted; the second field (resource) identifies a resource, as usual, in terms of a list of attributes; the last field (from) specifies which parties should grant resource access in exchange. The rule's owner party can specify that the resource access in exchange has to be granted to itself (me) or other parties and can specify as well that the resource access in exchange has to be provided by the party that sent the access request (requester) or other parties. The "other parties" can be identified using a list of attributes by selecting any (anySuchThat) or all (allSuchThat) parties in the policy system that satisfies them. Thus, exchanges can specify generic conditions of access to resources, not necessarily requiring access only for the rule's owner. As shown in the example below, the idea is that when evaluating a rule containing an exchange, the exchange will lead to the generation of subordinate requests.

A user *request* (resource : *Attributes*, from : *Others*) consists of a pair of labeled fields specifying the sequence of attributes that identify the resource to access (the field resource), and the target parties (the field from) which should provide the resource access.

The Bart evaluation process is triggered on demand to determine whether an access request can be granted based on the policies currently in force within the system. It is intrinsically context-aware, allowing authorization decisions to depend on dynamic, low-level data or environmental factors such as the current time, geopositioning data, and more. Contextual data is automatically retrieved as attributes during the evaluation process through a context handler. A *context* is represented as a list of attribute lists, where each list corresponds to the contextual attributes associated with a specific party within the policy system.

A collaborative delivery case study. We consider a case study where couriers affiliated with the same or different delivery companies collaborate by sharing information about addresses that are difficult to locate. When couriers manage to reach one of such addresses for the first time, they take note of the route to streamline future deliveries to that destination. This information can be invaluable to other couriers facing their first delivery to the same location. Despite potentially being competitors due to working for different companies, a collaborative approach benefits all parties by reducing time wastage and minimizing missed deliveries. At the same time, each courier retains autonomy over their strategy, deciding what information to share and what to ask in exchange.

Example 1: a simple two-courier scenario. Let us consider a simple scenario of the above case study that illustrates the basics of the Bart approach, with a particular focus on the resource exchange feature. The scenario involves two couriers affiliated with different companies whose collaboration is regulated by the policy system $PS_1 \triangleq R\ F$, where the policies R and F are defined as follows (hereafter, for the sake of presentation, we write $N \triangleq t$ to assign a name N to the term t):

$$\begin{aligned}
R \triangleq\ & (\mathsf{party} : (\mathit{service}\!:\!\mathit{delivery})(\mathit{company}\!:\!\mathit{RabbitService}),\\
& \quad \mathsf{rules} : (\mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Lucca})))\\
F \triangleq\ & (\mathsf{party} : (\mathit{service}\!:\!\mathit{delivery})(\mathit{company}\!:\!\mathit{FastAndFurious}),\\
& \quad \mathsf{rules} : (\mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Prato}),\\
& \qquad\qquad \mathsf{exchange} : (\mathsf{to} : \mathsf{me},\ \mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Pistoia}),\ \mathsf{from} :\ \mathsf{requester})\\
& \qquad\qquad\qquad\qquad \mathsf{or}\ (\mathsf{to} : \mathsf{me},\ \mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Lucca}),\ \mathsf{from} :\ \mathsf{requester})))
\end{aligned}$$

The policy R controls the accesses to resources of a courier that offers a *service* of type *delivery* and is affiliated to the *company* named *RabbitService*. The policy contains only one rule, which specifies the managed resource without imposing any condition for granting access and without requesting any resource in exchange. Specifically, the identified resource concerns information about difficult-to-find addresses (the resource *type* is *addrInfo*) located on the *city* of *Lucca*. The *RabbitService* courier makes her collected addresses available to everybody, thus implementing a very collaborative strategy.

The policy F controls the accesses to resources of a *FastAndFurious* courier. The policy contains only one rule, which refers to *addrInfo* concerning the *city* of *Prato*. This time, however, access to this resource is granted as long as the requester (of the *Prato* resource) gives in exchange to this courier (me) *addrInfo* concerning the *city* of *Pistoia* or *Lucca*.

For evaluating access requests, each party is univocally assigned an index. Thus, the *RabbitService* courier is assigned the party index 1, and the *FastAndFurious* courier is assigned the party index 2. A *user request* consists of a pair of labeled fields specifying the sequence of attributes that identify the resource to access (field resource) and the target parties (field from) which should provide the resource access. Target parties are specified using the syntax anySuchThat or allSuchThat described above.

Let us now consider the situation in which the *RabbitService* courier is making a delivery in *Prato*, and is having difficulty to find the delivery address. This courier can try to obtain information about addresses of *Prato* through the following (user) request:

$$\begin{aligned}
&(\mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Prato}),\\
&\ \mathsf{from} : (\mathsf{anySuchThat} :\ (\mathit{service}\!:\!\mathit{delivery})(\mathit{company}\!:\!\mathit{FastAndFurious})))
\end{aligned}$$

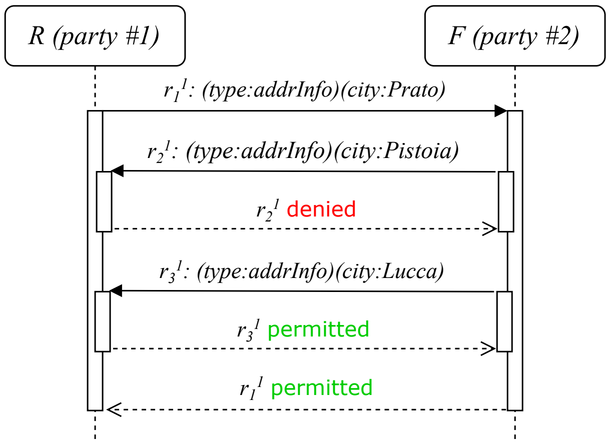
This user request, which contains the information explicitly specified by the user (i.e., the *RabbitService* courier), is enriched with the party index of the requester (i.e., 1), which is necessary for the evaluation. In addition, the from field of the request is evaluated concerning the attributes exposed in the party field of the policies, resulting in the set of indexes $\{2\}$ indicating the parties to which the request refers. We call such enriched requests, which are the ones that are effectively evaluated by our system, *point-to-point request*s. The resulting point-to-point request[1] is

$$r_1^1 \triangleq 1 : (\mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Prato}),\ \mathsf{from} :\ 2)$$

The request is then evaluated over the policy F of party 2, causing the generation and evaluation of subordinate requests due to the exchanges specified within F.

We depict the evaluation process for this scenario employing the sequence diagram reported in Fig. 1. Each point-to-point request is represented as a call message, depicted as a solid arrow from the requester to the resource holder. In contrast, the evaluation response is represented as a return message, depicted as a dashed arrow in the opposite direction. The processing of a request is graphically represented by activations (depicted as rectangles on the parties' lifelines). Notably, the arrows in the sequence diagram do not represent actual sent/received messages among the parties: they are only meant to represent

[1]Notably, we use r_i^j to denote the i-th point-to-point request of the j-th example.



R (party #1)
F (party #2)
r1^1: (type:addrInfo)(city:Prato)
r2^1: (type:addrInfo)(city:Pistoia)
r2^1 denied
r3^1: (type:addrInfo)(city:Lucca)
r3^1 permitted
r1^1 permitted

Figure 1: Request evaluations for the simple two-courier scenario.

the "steps" taken by Bart evaluation process, i.e., the generation of the subordinate requests during the evaluation of an exchange, and the access control decisions made following the specific requests.

Notice that requests generated by the evaluation of exchanges produce nested activations, which provide a graphical view of the level of request subordination. For the sake of presentation, we report the attributes identifying the requested resource for each request, and we highlight with green (resp. red) color the positive (resp. negative) evaluation results.

We comment on the salient points of the evaluation of request r_1^1. The request is evaluated on the F's rule for the *Prato* addresses. The rule applies to r_1^1, and the two exchanges (composed in or) are evaluated. The first exchange generates the subordinate point-to-point request

$$r_2^1 \triangleq 2 : (\mathsf{resource} : (\mathit{type}{:}\mathit{addrInfo})(\mathit{city}{:}\mathit{Pistoia}),\ \mathsf{from} :\ 1)$$

that is then evaluated on the policy R of party 1. This evaluation returns *false* (i.e., r_2^1 is denied) because the policy of the *RabbitService* courier does not provide any rule for the requested resource (i.e., addresses of *Pistoia*). The second exchange generates the subordinate point-to-point request

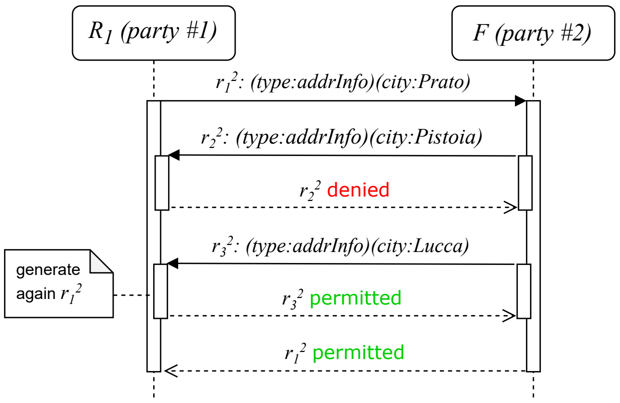
$$r_3^1 \triangleq 2 : (\mathsf{resource} : (\mathit{type}{:}\mathit{addrInfo})(\mathit{city}{:}\mathit{Lucca}),\ \mathsf{from} :\ 1)$$

that is evaluated on the policy R of party 1, again. This time, the evaluation returns *true* (i.e., r_3^1 is permitted) because the *RabbitService* courier provides access to this resource to everybody. Summing up, the two couriers reached the following agreement: the *FastAndFurious* courier provides access to information about *Prato*'s addresses to the *RabbitService* courier; in exchange, the latter provides access to information about *Lucca*'s addresses to the former.

3. Bart at Work

We show how the Bart approach applies to increasingly complex situations by considering different scenarios of our running case study.

Example 2: a two-courier scenario with a vicious exchange circle. This scenario illustrates how Bart's semantics deals with vicious circles of requests. As we saw in the scenario in Sec. 2, when the evaluation process encounters a rule that includes an exchange, additional subordinate requests are generated based on that exchange. In that scenario, these additional requests were satisfied by evaluating a rule without an exchange, allowing the evaluation to be completed successfully. However, when evaluating a request generated by an exchange, it is possible that one of the applied rules may itself contain another exchange, leading to yet another subordinate request. This can result in a nested



R_1 (party #1)
F (party #2)
r_1^2: (type:addrInfo)(city:Prato)
r_2^2: (type:addrInfo)(city:Pistoia)
r_2^2 denied
r_3^2: (type:addrInfo)(city:Lucca)
generate again r_1^2
r_3^2 permitted
r_1^2 permitted

Figure 2: Request evaluations for the two-courier scenario with a vicious exchange circle.

evaluation chain. During this chain, we might encounter a request that has already been processed earlier in the evaluation, but its evaluation is still pending. This creates a potential *vicious exchange circle*. In such cases, the conditions of the request under evaluation have already been validated (from when it was first encountered). Therefore, we can break the vicious circle[2] by terminating the evaluation of the current request and considering the evaluation chain a success, thanks to a mutual exchange. To avoid entering such vicious circles, we track the set of pending point-to-point requests that have already been encountered and whose conditions have been positively evaluated during the evaluation chain. Importantly, this set never includes requests that have been evaluated and denied.

The considered policy system now is $PS_2 \triangleq R_1\ F$, where F is defined as in Sec. 2 and R_1 is as follows:

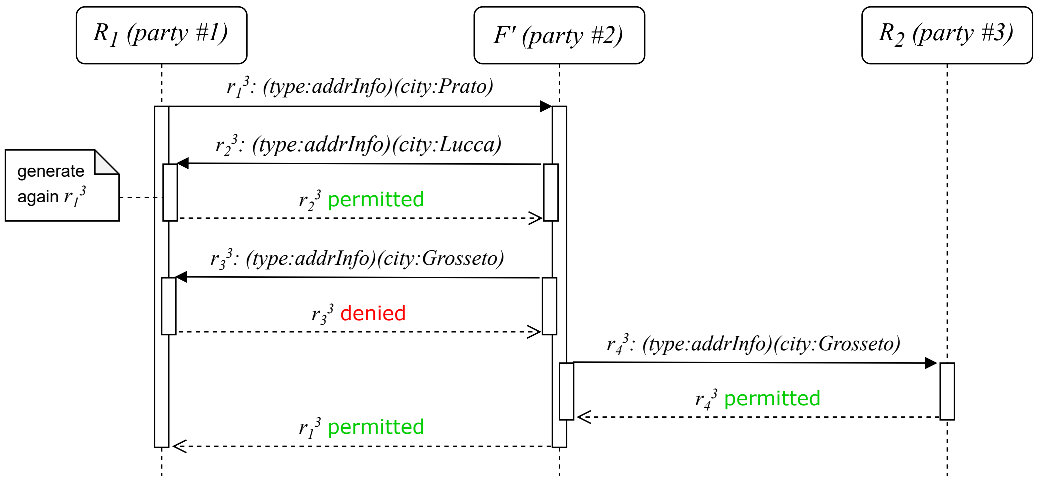
$$\begin{aligned}
R_1 \triangleq\ &(\mathsf{party} : (\mathit{service}\!:\!\mathit{delivery})(\mathit{company}\!:\!\mathit{RabbitService}),\\
&\quad \mathsf{rules} : (\mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Lucca}),\ \mathsf{condition} : \mathit{company} = \mathit{RabbitService})\\
&\qquad\qquad (\mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Lucca}),\ \mathsf{condition} : \mathsf{not}(\mathit{company} = \mathit{RabbitService}),\\
&\qquad\qquad\ \mathsf{exchange} : (\mathsf{to} : \mathsf{me},\ \mathsf{resource} : (\mathit{type}\!:\!\mathit{addrInfo})(\mathit{city}\!:\!\mathit{Prato}),\ \mathsf{from} :\ \mathsf{requester})))
\end{aligned}$$

This time, the *RabbitService* courier provides the addresses of *Lucca* without asking for anything in exchange to couriers of the same company. In contrast, the couriers of the other companies must give in exchange information about addresses of *Prato*.

Let us consider again the user request made by the *RabbitService* courier to obtain information about addresses of *Prato*. The request is authorized, provided that at least one of the subordinate requests r_2^2 and r_3^2 is authorized, as depicted in Fig. 2. Specifically, the evaluation of the first exchange of F generates the subordinate request r_2^2, which is denied (see the evaluation of request r_2^1 in Sec. 2). Instead, the evaluation of the second exchange of F generates the subordinate request r_3^2 that, in turn, through the exchange of R_1, generates a subordinate request r_4^2, which is equal to r_1^2 that belongs to the current set of pending requests $\{r_1^2\}$; thus, the evaluation of r_4^2 terminates, breaking the vicious circle. As a consequence, r_3^2 is permitted and, hence, the initial request r_1^2 is also permitted. Summing up, the two couriers reached the same agreement achieved in the scenario described in Sec. 2, i.e., the *FastAndFurious* courier provides access to information about *Prato*'s addresses to the *RabbitService* courier, in exchange of information about *Lucca*'s addresses.

Example 3: a multi-courier scenario. This scenario extends the previous one by considering multiple *RabbitService* couriers (for the sake of presentation, we consider two couriers, but the same policies work with a larger number). The scenario shows that the exchange of the *FastAndFurious* courier can be 'covered' by resources provided by different parties. The considered policy system is $PS_3 \triangleq R_1\ F'\ R_2$, where R_1 is like in the previous scenario, and the other policies are as follows:

[2]In the final example of this section, we will demonstrate how we can further relax the condition for detecting a vicious circle and still terminate the evaluation successfully.



R_1 (party #1)
F' (party #2)
R_2 (party #3)
r_1^3: (type:addrInfo)(city:Prato)
r_2^3: (type:addrInfo)(city:Lucca)
generate again r_1^3
r_2^3 permitted
r_3^3: (type:addrInfo)(city:Grosseto)
r_3^3 denied
r_4^3: (type:addrInfo)(city:Grosseto)
r_4^3 permitted
r_1^3 permitted

Figure 3: Request evaluations for the multi-courier scenario.

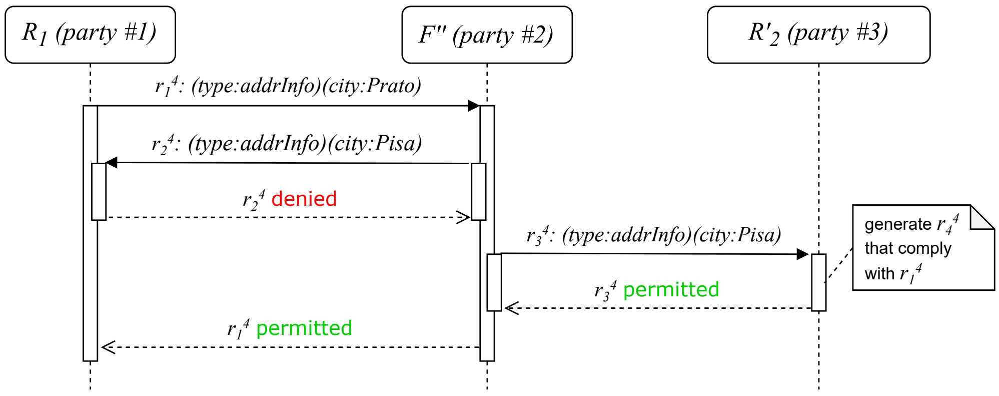
$$\begin{aligned}
F' \triangleq\ & (\mathsf{party} : (\mathit{service}{:}\,\mathit{delivery})(\mathit{company}{:}\,\mathit{FastAndFurious}),\\
& \quad \mathsf{rules} : (\mathsf{resource} : (\mathit{type}{:}\,\mathit{addrInfo})(\mathit{city}{:}\,\mathit{Prato}),\\
& \qquad \mathsf{exchange} : (\mathsf{to} : \mathsf{me},\ \mathsf{resource} : (\mathit{type}{:}\,\mathit{addrInfo})(\mathit{city}{:}\,\mathit{Lucca}),\\
& \qquad\qquad \mathsf{from} :\ \mathsf{anySuchThat} : (\mathit{service}{:}\,\mathit{delivery})(\mathit{company}{:}\,\mathit{RabbitService}))\\
& \qquad\qquad \mathsf{and}\\
& \qquad\qquad (\mathsf{to} : \mathsf{me},\ \mathsf{resource} {:} (\mathit{type}{:}\,\mathit{addrInfo})(\mathit{city}{:}\,\mathit{Grosseto}),\\
& \qquad\qquad \mathsf{from} :\ \mathsf{anySuchThat} : (\mathit{service}{:}\,\mathit{delivery})(\mathit{company}{:}\,\mathit{RabbitService}))))\\
R_2 \triangleq\ & (\mathsf{party} : (\mathit{service}{:}\,\mathit{delivery})(\mathit{company}{:}\,\mathit{RabbitService}),\\
& \quad \mathsf{rules} : (\mathsf{resource} : (\mathit{type}{:}\,\mathit{addrInfo})(\mathit{city}{:}\,\mathit{Grosseto})))
\end{aligned}$$

This time, the *FastAndFurious* courier offers the *Prato* addresses in exchange for addresses of both *Lucca* and *Grosseto*. Differently from the previous scenario, the policy F' does not impose that the data in exchange is provided by the courier requesting the *Prato* addresses: it is enough that the data is provided by any *RabbitService* courier.

The interactions among the parties are depicted in Fig. 3. We comment on the salient points. The *RabbitService* courier with policy R_1 sends the usual request concerning the *Prato* addresses (request r_1^3). The *FastAndFurious* courier reacts by requesting access to addresses of *Lucca* and *Grosseto* to the requesting *RabbitService* courier (requests r_2^3 and r_3^3). The former request is permitted (the vicious circle generated by the exchange within the policy R_1 is broken as in the previous scenario). In contrast, the latter request is denied because the party does not have the resource corresponding to the *Grosseto* addresses. Thus, the *FastAndFurious* courier contacts the other *RabbitService* courier (request r_4^3), as prescribed by the use of anySuchThat in the from field of the F''s exchange. This request is permitted (since R_2 offers access to this resource without restriction). Therefore, the exchange of F' is satisfied; hence, the initial request r_1^3 is permitted.

Example 4: a more complex exchange relationships scenario. This scenario underscores the importance of the context handler in evaluating policies that depend on environmental attributes, such as the current position and time of the requesting courier. Additionally, the scenario demonstrates how a third party can resolve a vicious circle involving two parties by using anySuchThat in the to field of the exchange. Finally, it illustrates that detecting vicious circles arising from requests generated by exchanges does not require an exact match between the newly generated subordinate request and one of the pending requests. Instead, it suffices that the new request is less demanding than the pending one—for example when it involves the same parties but fewer attributes. In such cases, the evaluation process breaks the vicious circle and successfully resolves the original request.

The considered policy system is $PS_4 \triangleq R_1\ F''\ R_2'$, where R_1 is like in the previous scenarios, and the



R_1 (party #1)
F'' (party #2)
R'_2 (party #3)
r_1^4: (type:addrInfo)(city:Prato)
r_2^4: (type:addrInfo)(city:Pisa)
r_2^4 denied
r_3^4: (type:addrInfo)(city:Pisa)
generate r_4^4 that comply with r_1^4
r_3^4 permitted
r_1^4 permitted

Figure 4: Request evaluations for the complex scenario.

other policies are as follows:

$$\begin{aligned}
F'' \triangleq\ & (\mathsf{party} : (service\!:\!delivery)(company\!:\!FastAndFurious),\\
&\quad \mathsf{rules} : (\mathsf{resource} : (type\!:\!addrInfo)(city\!:\!Prato),\\
&\qquad\quad \mathsf{condition} : timeH > 07{:}00 \text{ and } timeH < 20{:}00 \text{ and } position = Prato\\
&\qquad\quad \mathsf{exchange} : (\mathsf{to} : \mathsf{me},\ \mathsf{resource} : (type\!:\!addrInfo)(city\!:\!Pisa),\\
&\qquad\qquad\qquad \mathsf{from} :\ \mathsf{anySuchThat} : (service\!:\!delivery)(company\!:\!RabbitService))))\\
R'_2 \triangleq\ & (\mathsf{party} : (service\!:\!delivery)(company\!:\!RabbitService),\\
&\quad \mathsf{rules} : (\mathsf{resource} : (type\!:\!addrInfo)(city\!:\!Grosseto))\\
&\qquad\quad (\mathsf{resource} : (type\!:\!addrInfo)(city\!:\!Pisa)\\
&\qquad\quad \mathsf{exchange} : (\mathsf{to} : \mathsf{anySuchThat} : (service\!:\!delivery)(company\!:\!RabbitService),\\
&\qquad\qquad\qquad \mathsf{resource} : (type\!:\!addrInfo),\ \mathsf{from} :\ \mathsf{requester})))
\end{aligned}$$

In this scenario, the *FastAndFurious* courier provides the addresses of *Prato* provided that the requester currently needs that information: the rule's condition checks that the requester is positioned in *Prato* and it is a working hour (between 7:00am to 8:00pm). The addresses of *Pisa* are asked in exchange. The policy of the second *RabbitService* courier is extended with a rule controlling the access to the information concerning the *Pisa*'s addresses. The courier grants access to this information provided that the requester provides any address information to any courier of the *RabbitService* company.

Let us consider the usual request of *Prato* addresses, evaluated with respect to the context

$$((timeH\!:\!10{:}00)(position\!:\!Prato))\ ((timeH\!:\!10{:}00)(position\!:\!Prato))\ ((timeH\!:\!10{:}00)(position\!:\!Pisa))$$

specifying that the current time is 10:00 am, and the first *RabbitService* courier and the *FastAndFurious* courier are located at *Prato*, while the second *RabbitService* courier is located at *Pisa*.

The interactions among the parties are depicted in Fig. 4. We comment on the salient points. The *RabbitService* courier with policy R_1 sends the initial request (r_1^4). The *FastAndFurious* courier reacts by requesting access to addresses of *Pisa* to the requesting *RabbitService* courier (request r_2^4). This request is denied. Thus, the same resource is requested to the other *RabbitService* courier (request r_3^4). The evaluation of this request involves the exchange within the policy R'_2, which generates subordinate requests directed to the *FastAndFurious* courier from any *RabbitService* courier for any address information. In the figure, the first generated request is $r_4^4 \triangleq 1 : (\mathsf{resource} : (type\!:\!addrInfo),\ \mathsf{from} :\ 2)$. This request involves the same parties of the original request r_1^4, which is recorded in the set of pending requests. Request r_4^4 is "weaker" than r_1^4: it requests only the attribute $(type\!:\!addrInfo)$, while r_1^4 also requests $(city\!:\!Prato)$. Nevertheless, as anticipated, the evaluation process also breaks the vicious circle in this case and considers the evaluation a success. In fact, in this scenario, it is perfectly reasonable to positively evaluate the exchange within the policy R'_2 because it requires that a *RabbitService* courier obtains access to some address information from the *FastAndFurious* courier and, indeed, this requirement is satisfied by the initial request that grants access to a *RabbitService* courier to *Prato* addresses provided by the *FastAndFurious* courier. Thus, request r_3^4 is permitted and, hence, request r_1^4 is also permitted.

Notably, the evaluation of the exchange of R'_2 may generate $3 : (\text{resource} : (type\!:\!addrInfo),\ \text{from} :\ 2)$ as the first subordinate request; however, it would be denied because the condition of the *Prato*'s rule within F'' would not be satisfied, as the second *RabbitService* courier is positioned at *Pisa* and not *Prato*. Thus, also in such a case, the final permitted requests would be r_3^4 and r_1^4.

4. Related Work

An overview of the literature on access control for collaborative systems and the associated tools can be found in, e.g., [8, 9]. However, to the best of our knowledge, the topic of bartering resource access has not yet been thoroughly investigated. Below, we discuss the most strictly related work.

In [10], the authors present *MuAC*, a logically-based access control language tailored for expressing the exchange of access rights in scenarios such as data sharing on social networks. Beyond traditional access control requirements, *MuAC* enables the specification of conditions on what requesters must offer in return for accessing a particular resource. To determine if a request can be granted and to address potential circularities arising from mutual access policies, *MuAC* relies on the PCL (Propositional Contract Logic [11]) proof system. This is achieved through the translation of *MuAC* policies and requests into a finite set of PCL propositions as a preventive step. In contrast, Bart adopts a more flexible, attribute-based approach for identifying subjects and resources and defining policies. Additionally, it employs a more operational mechanism for evaluating request grantability and resolving circularities.

Another approach to modeling resource exchange is proposed in [12]. The authors, some of whom also contributed to [10], introduce the concept of an *exchange environment*, a type of online platform where users can exchange resources. They present a logical language, also named MuAC, for declaratively defining users' policies, along with a mechanism to ensure that resource exchange is *fair*. Fairness, in this context, means adhering to all involved users' policies while preventing issues like double spending. A key distinction between this approach and Bart lies in the nature of the resources exchanged. In [12], the focus is on consumable resources whose ownership is transferred during the exchange. In contrast, Bart exchanges access rights to resources rather than their ownership. This ensures that the resource owner retains the ability to modify access control policies at any time.

In [13], the authors propose using a new permission, called *mutual*, in addition to the classical ones permit and deny. By exploiting this permission, users can write policies for granting access to their resources only to users who allow them access to theirs. The authors define the syntax and semantics of mutual authorization policies and extend the RBAC model accordingly. Our approach is more expressive in that Bart's syntax allows users to explicitly specify which other resources and parties should be involved in an exchange, while in [13], the involved resources and parties are implicitly identified on the base of the request.

In [14], a novel policy-based access control model is proposed for collaborative environments. Access control decisions become a collaborative activity in which a set of collaborating parties must enforce a global policy without compromising their autonomy or confidentiality requirements. To this aim, the global policy is decomposed into local policies so that the policies local to a party only need the information available at that party and that the decisions obtained from the local policies can be combined to derive the access control decision for the collaboration as a whole. In [15], the authors propose an extension of XACML specifically developed for distributed systems in which the involved parties can autonomously specify their access control policies and might not be willing to share or lease the ownership of their resources. The extension allows a party to specify the approach to be taken if her policies have to be integrated with others. Although the goals of these two papers differ from ours, our approach has in common the basic idea that, when evaluating a request, the final decision is obtained by combining the results of evaluating policies specified by different users.

In a sense, the Bart's evaluation process shares similar objectives with so-called *fair exchange protocols*, i.e., protocols ensuring that no party in a transaction can gain an advantage over the other parties by misbehaving, cheating, or prematurely aborting the protocol (see [16] for a survey). The difference lies in the way in which the objective is achieved, as in Bart it is the evaluation process that implicitly executes a sort of protocol for assuring that a requested resource access complies with (the conditions specified in) the parties' policies.

Finally, in [17], the authors propose an attribute-based controlled collaborative access control scheme for public cloud storage. The aim is to ensure data confidentiality and collaborative access control by grouping users so that only users involved in the same project can collaborate. Instead, we aim to enable different users to exchange access to resources without forcing them to be part of specific groups.

5. Concluding Remarks and Future Work

In this paper, we have proposed a framework for expressing collaborative policies indicating conditions that must be satisfied by system parties to barter access to their resources. This materializes in Bart, a novel policy language equipped with a dedicated exchange construct enabling reaching agreement on resource access. The interactions among parties are made flexible and expressive by allowing requests and exchanges to address groups of parties, selecting them according to their attribute values. Bart's evaluation process ensures faithful enforcement of the parties' policies while managing the decision-making process to guarantee compliance. The implementation of Bart as a Java library is currently under development.

We defined the language Bart with a minimal set of constructs in order to focus on the collaborative aspects of policies, thus enabling a formal definition of the mechanisms supporting the authorization of resource exchanges. As a future development, we intend to apply the same approach to full-fledged policy languages, such as the formal language FACPL [18] and the OASIS standard XACML [6]. Regarding the linguistic features for specifying collaborative policies, we plan to extend the range of conditions a party can define in their policies. For instance, a party could specify a condition requiring the requester to commit not to share their resources with certain specified parties in exchange for access to the party's resources. Another compelling condition could involve the requester sharing, or having previously shared, a resource with another specific party or a group of system participants. Moreover, we aim to explore methods for governing runtime changes to policies. This is particularly crucial in our context, as a malicious party could exploit temporary policy changes to gain access without effectively providing the opportunity for other parties to get something in exchange. Finally, we plan to study general results (concerning, e.g., information flow and reasonability properties) and policies' specific properties.

Declaration on Generative AI

The authors have not employed any Generative AI tools.

References

[1] B. W. Lampson, Protection, Operating Systems Review 8 (1974) 18–24.

[2] D. F. Ferraiolo, D. R. Kuhn, Role-based access control, in: NIST-NCSC National Computer Security Conference, 1992, pp. 554–563.

[3] V. C. Hu, D. R. Kuhn, D. F. Ferraiolo, Attribute-based access control, IEEE Computer 48 (2015) 85–88.

[4] X. Jin, R. Krishnan, R. S. Sandhu, A unified attribute-based access control model covering DAC, MAC and RBAC, in: DBSec, Springer, 2012, pp. 41–55.

[5] NIST, A survey of access control models, 2009. http://csrc.nist.gov/news_events/privilege-management-workshop/PvM-Model-Survey-Aug26-2009.pdf.

[6] OASIS XACML TC, eXtensible Access Control Markup Language (XACML) version 3.0 , 2013. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[7] L. Bettini, R. Pugliese, F. Tiezzi, Bart's syntax and semantics, Technical Report, Università degli Studi di Firenze, 2024. Available at https://github.com/LorenzoBettini/bart-syntax-semantics/blob/master/bart_formalization.pdf.

[8] F. Paci, A. Squicciarini, N. Zannone, Survey on Access Control for Community-Centered Collaborative Systems, ACM Computing Surveys 51 (2018) 1–38.

[9] R. Fernandez, P. C.-H. Cheng, A. Nhlabatsi, K. M. Khan, N. Fetais, Effective Collaboration in the Management of Access Control Policies: A Survey of Tools, IEEE Access 11 (2023) 13929–13947.

[10] L. Ceragioli, P. Degano, L. Galletta, MuAC: Access Control Language for Mutual Benefits, in: CEUR WORKSHOP PROCEEDINGS, volume 2597, 2020, pp. 119–127.

[11] M. Bartoletti, R. Zunino, A calculus of contracting processes, in: Proceedings of the 2010 25th Annual IEEE Symposium on Logic in Computer Science, LICS '10, IEEE Computer Society, 2010, p. 332–341.

[12] L. Ceragioli, L. Galletta, P. Degano, L. Viganò, Policies for Fair Exchanges of Resources, arXiv preprint arXiv.2410.21214 (2024).

[13] G. Suntaxi, A. A. E. Ghazi, K. Böhm, Mutual Authorizations: Semantics and Integration Issues, in: Proceedings of the Symposium on Access Control Models and Technologies, ACM, 2019, pp. 213–218.

[14] D. Lin, P. Rao, E. Bertino, N. Li, J. Lobo, Policy Decomposition for Collaborative Access Control, in: ACM Symposium on Access Control Models and Technologies, ACM, 2008, pp. 103–112.

[15] P. Mazzoleni, B. Crispo, S. Sivasubramanian, E. Bertino, XACML policy integration algorithms, ACM Trans. Inf. Syst. Secur. 11 (2008) 4:1–4:29.

[16] I. Ray, I. Ray, Fair Exchange in E-commerce, ACM SIGecom Exchanges 3 (2002) 9–17.

[17] Y. Xue, K. Xue, N. Gai, J. Hong, D. S. Wei, P. Hong, An Attribute-Based Controlled Collaborative Access Control Scheme for Public Cloud Storage, IEEE Transactions on Information Forensics and Security 14 (2019) 2927–2942.

[18] A. Margheri, M. Masi, R. Pugliese, F. Tiezzi, A Rigorous Framework for Specification, Analysis and Enforcement of Access Control Policies, IEEE Trans. Software Eng. 45 (2019) 2–33.