

Control your eLearning environment. Exploiting policies in an open infrastructure for lifelong learning

Juri L. De Coi, Philipp Kärger, Arne W. Koesling, and Daniel Olmedilla

Abstract—Nowadays, people are in need for continuous learning in order to keep up to date or to be upgraded in their job. An infrastructure for life-long learning requires continuous adaptation to learners' needs and must also provide flexible ways for students to use and personalize them. Controlling who can access a document, specifying when a student may be contacted for interactive instant messaging or periodical reminders in order to increase motivation for collaboration are just some examples of typical statements that may be specified by e.g., learners and learning management system administrators. This paper investigates how existing work in the area of policy representation and reasoning can be used in order to express these statements while at the same time obtaining the extra benefits policies provide (e.g., flexibility, dynamicity and interoperability). The paper analyzes existing policy languages and integrates one of them as part of a demonstration of its feasibility in providing more advanced and flexible eLearning environments.

Index Terms—policies, eLearning, life-long learning, PROTUNE, rule, reactivity

I. INTRODUCTION

Society and current labor market evolve rapidly. Nowadays, a learner is potentially any person in the world, who wants to learn or keep up to date on any specific topic, be it at work or in any other facet of her life. Therefore, there is a growing need for more flexible and cost-effective solutions allowing learners to study at different locations (e.g., at home) and at times that are better arranged with their working hours. In addition, learners do not necessarily work isolated but may collaborate with or contact other persons, learners or tutors. Systems addressing these requirements must allow users to have a big flexibility in the way they use the system, how they collaborate, how they share their content, etc. Controlling who can access a document, specifying when a student may be contacted for interactive instant messaging or periodical reminders in order to increase motivation for collaboration are just some of the examples of typical statements that may be specified, for instance, by learners and learning management system administrators.

Research performed in the area of policy representation and reasoning allows for very expressive languages in order to specify statements that learners, course designers or administrators can use to enhance their interactions with learning agents and management systems. Furthermore, lately there has been extensive research that provides not only the ability of specifying these

statements but also advanced mechanisms for reasoning over, exchanging and exploiting them [1]–[7]. This paper focuses on the use of policies, a well-defined flexible and dynamic approach in order to specify and control the behavior of complex and rapidly evolving infrastructures for life-long learning. It also explores how the integration of a policy framework can increase the flexibility of the interactions and collaborations learners have with learning agents and management systems, therefore enhancing their experiences and learning. The work presented in this paper builds on [8] and adds the following contributions:

- More detailed scenario and analysis of requirements
- Extended comparison among existing policy frameworks
- Description of the syntax and semantics of the PROTUNE framework as well as its architecture
- Integration of the PROTUNE framework into a web-based demonstration of the scenarios described in the paper
- Experimental results on performance of the policy evaluation process

The rest of the paper is structured as follows. First, Section II identifies example situations in which the specification of policies would increase the flexibility of the interactions and collaborations as well as enhance the learners experience. These examples show that dynamicity and ease of use are a crucial requirement, both being two of the main characteristics of policies. An introduction into the area of policy representation and reasoning, including a definition of the term policy as well as the characteristics of policies, is provided in Section III. The benefits of the integration of policies into learning management systems and personal learner agents in order to support advanced scenarios are described in Section IV, as well as the out-of-the-box benefits of their exploitation. In addition, Section V analyzes existing policy languages and frameworks in order to present an overview of available solutions to the reader. It provides a comparison of their main features as well as their advantages and disadvantages from the perspective of their integration into lifelong learning infrastructures. It also introduces the formalization of policies using a selected policy language (PROTUNE) and describes some of the added benefits of its use, such as negotiations and advanced explanations. The architecture of the selected policy framework, its integration into an on-line demonstration as well as a performance evaluation is presented in Section VI. Finally, related work is presented in Section VII and Section VIII concludes the paper.

II. MOTIVATION SCENARIO

Alice holds a master degree in computer science and works successfully in a company. Recently, Alice was assigned the task of managing a new project starting in a couple of months and

Authors order is alphabetical.

Authors work at L3S Research Center & Leibniz University of Hanover, Hanover, Germany. Arne Koesling also works at the Hannover Medical School, Hanover

E-mail: {decoi,kaerger,koesling,olmedilla}@L3S.de

therefore she would need to learn and refresh her knowledge on project management. Since she has a full time job including many business trips, she uses an on-line learning client that allows her to improve her competences whenever she has some available time. With this learning client she is able to collaborate and to send questions or answers to other learners or tutors, and therefore she is able to chat with other students and even participate in a social network. However, since she uses her chat tool also for her job she restricts her chat facility in a way that during working time only business contacts and other employees of her company can start a conversation, therefore, allowing other students to contact her only in her leisure time. Of course, students trying to contact her during working time get a brief explanation of why a conversation is not possible at that very moment and which even indicates when Alice can be contacted.

Within the program Alice is following, she accesses different learning activities and objects through her learning client. Some of this material is free of charge, but a couple of learning activities she is interested in are offered each one by a different content provider that sells it. Since the material is sold at a good price, she decides to purchase it. Each provider tells Alice that either she has to have an account or she has to provide a credit card for payment of the learning activity. For the first provider she does have an account and provides her user name and password. Therefore, she retrieves the requested material. However, she does not know the second provider and she must disclose her credit card. Alice protected her credit card in a way that it would only be disclosed to providers she may trust and the learning client provides a mechanism by which a content provider and Alice can trust each other even if they have not had any transaction in common before.

The learning client Alice is using allows her to share exercises and other relevant documents stored in her computer (e.g., using a peer-to-peer network [9], [10] or uploading them to a server) with other students following the same program or within the same learning network. She may even create some new material out of what she learned and her experience at work. She specifies which documents are to be shared and which conditions other students must fulfill in order to be able to retrieve it (e.g., being part of the same program she is enrolled or being a tutor). Even if life-long learning means that you can learn whenever you like, the human factor still plays an important role in the setting, regarding motivation and exchange of information. Therefore, inactivity may yield the danger of not keeping up with her learning group. In order to ensure the success of the students, the learning client includes a personalizable agent. Among other uses for this agent, Alice can create some guidelines in a way that the agent reminds her when she has to finish some learning activities or sends her an e-mail when she has been inactive for more than a week.

Bill is a tutor and on-line course designer for the university, in which Alice attends many of her on-line courses with the help of her learning client. In his role as tutor, Bill specifies in the system that any of his learners not having any activity during more than two weeks, should be sent a message or notification asking whether she needs additional help.

Bill, in his role of course designer, could also specify in the courses he creates, that some parts of the course are shown with more or less information based on the learner accessing it (e.g., whether she has already tried the formative assessment of the course) or even dynamically link to different kind of contents

based on the information the learner provides dynamically at the time she is accessing the course (e.g., provide on-line games [11] in case the learner notifies she prefers [12] those kind of learning resources).

Thanks to all these flexible facilities and all their personalization and configuration possibilities, Alice is able to finish her program successfully.

III. POLICIES—A BRIEF INTRODUCTION

This section briefly introduces what the term policy refers to and the advantages of policies with respect to more conventional approaches. It also describes the features a policy framework would ideally provide, which will be used in Section V as part of the comparison criteria, in order to select the framework that better meets the requirements of a life-long learning scenario such as the one presented previously.

A. The concept of policy

The term policy can be generally defined as a “statement specifying the behavior of a system”, i.e., a statement which describes which decision the system should take or which actions it should perform according to specific circumstances. Policies are encountered in many situations of our daily life: the following example is an extract of a return policy of an on-line shop:¹

Any item for return must be received back in its original shipped condition and original packing. The item must be without damage or use and in a suitable condition for resale. All original packaging should accompany any returned item. We cannot accept returns for exchange or refund if such items have been opened from a sealed package.

With the digital era, the specification of policies has emerged in many web-related contexts and software systems. E-mail client filters are a typical example of policies. The following policy is an example for an e-mail filter addressing spam:

If the header of an incoming message contains a field “X-Spam-Flag” whose value is “YES” then move the message into the folder “INBOX.Spam”. Moreover if this rule matches, do not check any other rules after it.

Some of the main application areas where policies have been lately used are security and privacy as well as specific business domains (where the type of policies used are usually called “business rules”). A security policy defines security restrictions for a system, organization or any other entity. It may define which resources a system should regard as security relevant, in which way the resources should be protected and how the system should proceed, if access to those resources is requested by a third party. A privacy policy is a declaration made by an organization regarding its use of customers’ personal information (e.g., whether third parties may have access to customer data and how that data will be used). Finally, business rules describe the operations, definitions and constraints that apply to an organization in achieving its goals. Table I presents extracts of a security policy², a privacy policy³ and a business rule⁴ (in the specific business domain of a tax

¹taken from <http://www.cheeki-monkey.co.uk/shopcontent.asp?type=returnspolicy>

²http://www.sans.org/resources/policies/DMZ\Lab\Security_Policy.doc

³http://www.siteprocentral.com/contracts/privacy_policy_sample.html

⁴<http://businessrules.brmsblog.com/2005/08/10/examples-of-business-rulessee-my-previous-post/>

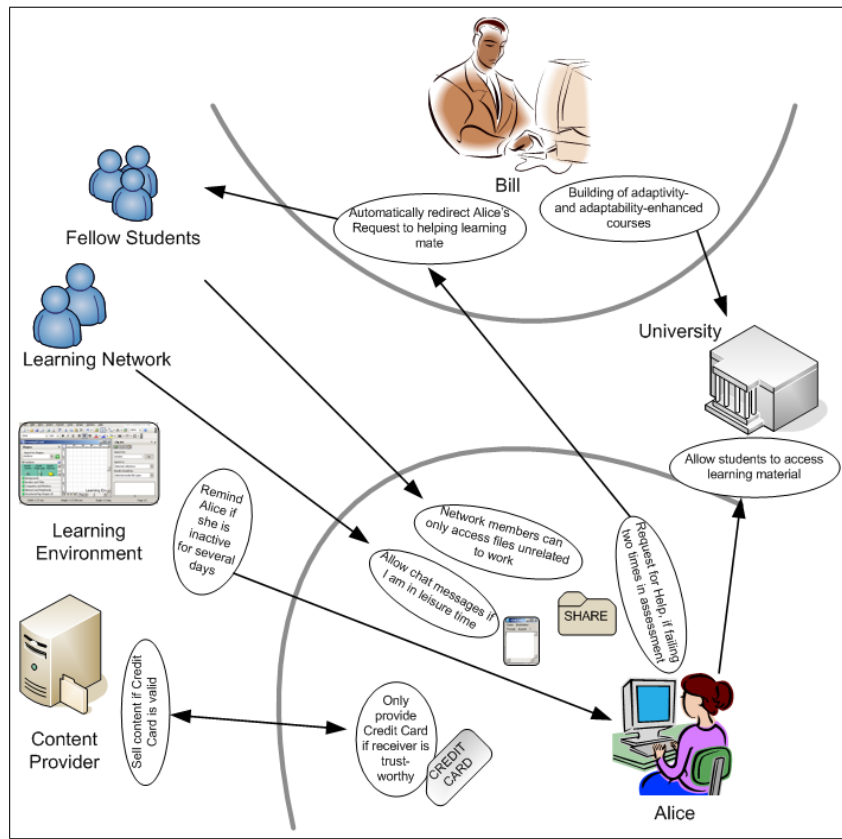


Fig. 1. Example policies in an open and flexible life-long learning infrastructure

(1) Original firewall configurations and any changes thereto must be reviewed and approved by InfoSec (including both general configurations and rule sets). InfoSec may require additional security measures as needed.
(2) Our site users can choose to electronically forward a link, page, or documents to someone else by clicking "e-mail this to a friend". The user must provide their email address, as well as that of the recipient. This information is used only in the case of transmission errors and, of course, to let the recipient know who sent the email. The information is not used for any other purpose.
(3) A person is entitled to an exemption from taxation of the tangible personal property the person owns that is held or used for the production of income if that property has a taxable value of less than \$500.

TABLE I
EXAMPLES FOR POLICIES: (1) A SECURITY POLICY, (2) A PRIVACY POLICY, (3) A BUSINESS RULE

collection agency).

B. Advantages of policies

Specification of policies using a policy language yield many advantages compared to other conventional approaches: they are dynamic, typically declarative, have normally well-defined semantics and usually allow for reasoning over them. In the following all above-mentioned policy properties will be thoroughly described.

a) *dynamic*: The description of the behavior of an agent or other software component is usually built in the component itself. The main drawback of this design choice is that whenever the need for a different behavior arises and new code for that

behavior is created, it typically requires the re-compilation and re-installation (or update) of the software. A more reusable design choice should provide a component with the ability of adapting its behavior according to some dynamically configurable description of the desired behavior. In this case, as soon as the need for a different behavior arises, only the description of the behavior will need to be replaced and not the whole component. Being policies, as mentioned above, "statements specifying the behavior of a system", in order to change the behavior of a policy engine (i.e., a component able to enforce policies) simply replacing the old policy with the new one would be enough.

b) *declarative*: The traditional (*imperative*) programming paradigm requires programmers to explicitly specify an algorithm to achieve a goal. On the other hand the *declarative* approach simply requires that programmers specify the goal, whereas the implementation of the algorithm is left to the support engine. This difference is commonly expressed by resorting to the sentence "declarative programs specify *what* to do, whereas imperative programs specify *how* to do it". For this reason declarative languages are commonly considered a step closer to the final user than imperative ones. Policy languages are typically declarative and policies are typically declarative statements and as such they can be more easily defined by final (possibly non-computer experts) users. The policies listed in Table I are declarative as well: for instance, the first one does not explain which steps the process of reviewing and approving a firewall configuration consists of, but simply asserts under which circumstances they have to be reviewed and approved.

c) *well-defined semantics*: A language's semantics is well-defined if the meaning of a program written in that language

is independent of the particular implementation of the language. Logic programs and Description Logic knowledge bases have a mathematically defined semantics, therefore, we assume languages based on either of the two formalisms to have well-defined semantics. Programs written in a language provided with a well-defined semantics are easily exchangeable among different parties since each party understand them in the same way. On the other hand, natural languages sentences are ambiguous and can be interpreted differently. Policies with well-defined semantics, therefore, have advantages over policies written in a natural language as the ones provided in Table I.

d) reasoning: The term “reasoning” refers to the possibility of combining known information in order to infer new one, like in the following example.

If it is known that “all humans are mortal” and that “Socrates is human” one can infer that “Socrates is mortal”.

On the one hand it is true that the sentence “Socrates is mortal” is different than the ones preceding it, but on the other one it is clear that, according to the common sense, one can deduce (i.e., *infer*) the third sentence from the first two. The inferred information is referred to as *implicit* knowledge, since it was not explicitly available before. In the context of declarative programs, statements a program consists of can be reasoned over in order to infer new statements. Reasoning applied to the third policy in Table I allows to deduce that (for instance) if

- John is a person,
- John owns a car,
- the car is a tangible personal property,
- John uses the car in his daily work (and therefore, he uses it for the production of income), and
- the car has a taxable value of less than \$500

then John is entitled to an exemption from taxation of the car.

C. *Desiderata of policy languages*

Formally specified policies generally have the features described in the previous section. Nevertheless, current policy languages assume that the frameworks enforcing policies written in such policy languages support more advanced features. A list of such additional features is given next:

- **Positive vs. negative authorization:** policies specifying conditions under which resources can be accessed may be of two types: positive or negative. Positive authorization policies specify that if the conditions are satisfied, some authorization is granted, e.g., “access is granted if the requester is a member of the company”. Whereas negative authorization policies specify that if the conditions are satisfied some authorization is denied, e.g., “access is denied if the requester is a member of a foreign company”. Positive/Negative policies retain the natural way people express policies, nevertheless, it can be argued that the specification of negative authorizations complicates the enforcement of access control in a system [3] and comprises the extra complexity of having to deal with conflicts. A conflict situation arises, whenever there are policies applicable for the same situation: one granting authorization and the other denying it. In the context of security and access control, a typical approach is assuming that access to any resource is denied by default and only positive authorization policies

are defined stating which resources are allowed [2], [4]. The reason is that the cost of disclosing a sensitive resource is much higher than the cost of not disclosing a non-sensitive one. However, for frameworks where these conflicts may arise, different conflict resolution strategies are provided [1], [5] (e.g., static detection at the specification time or at run-time with precedence metapolicies).

- **Negotiations:** In traditional access control or authorization scenarios only one party is able to specify policies which the other one has to conform to. Typically only one of the interacting parties is enabled to specify the requirements the other has to fulfill, whereas the other has no other choice but satisfying them (and thereby being authorized) or not (and thereby not being authorized). An example for this classical approach is the payment process in current on-line stores: the shop specifies what a customer has to provide (e.g., a credit card number) in order to purchase a product, but this is a one-way street: the customer has no means to require some constraints the shop has to satisfy too (e.g., trust evidences of the shop). Therefore, a more expressive approach allows both parties to discuss (i.e., negotiate) in order to reach an agreement. In an on-line buying process both the selling and the buying parties want the transaction to be successful (because both parties can take advantage from a successful transaction) and therefore are willing to make every possible effort leading to a successful transaction. This kind of transaction may require support for policy-driven negotiations. Furthermore, negotiations allow for some policies to be private, possibly being dynamically disclosed to other parties based on the satisfaction of some conditions.
- **Evaluation and Actions:** The evaluation of a policy is a process that checks whether it is satisfied or not (that is, whether it holds or not). The infrastructure enforcing a policy is in charge of its evaluation. Typically, a request or an event causes a policy to be evaluated in order to check what kind of behavior (e.g., grant access, move to spam folder, etc.) has been triggered. In order to evaluate a policy, some actions performed by the policy infrastructure might be required. Examples are a query to legacy systems (e.g., a database storing who is member of a given company) or other sources (e.g., the checking of a credit card’s validity at an external web service), and the sending of evidences (e.g., certificates, digital driver’s license) in order to certify some properties of a party. Another common action is the retrieval of environmental properties like the current system time (e.g., if access is allowed only in a specific time frame) or location (e.g., share files with learners in the same meeting room).
- **Explanations:** It should be possible to generate explanations [13] out of the policies and the decisions they make. On the one hand, they help a user to check whether the policies she created are correct and, on the other hand, they inform other users about why a decision was taken (or how the users can change the decision by performing a specific action). For example, if a student tries to contact Alice during her working time, that student would rather appreciate receiving a message like “I am not available from 8:00am to 5:00pm” instead of “I am not available”. Or if Alice discloses her credit card number to a content provider and it is not accepted, a message like “This credit card is invalid because

it is expired” would be more useful than simply “Invalid credit card”.

- **Strong/lightweight evidences:** The result of a policy’s evaluation may depend on the identity or other properties of a requester such as age, membership in a certain club, etc. Therefore, a policy language should provide a means to communicate such properties. Usually, this is done by sending digital certificates called *strong evidences*. Typical strong evidences are credentials signed by trusted entities called *certification authorities*. *Lightweight evidences* [14] are non signed declarations or statements (e.g., license agreements). As an example, the driver’s license number maintained as an integer value is a lightweight evidence. A digital version of the driver’s license that can be submitted to a certification authority in order to prove that it has been signed by the government and that contains certified properties (e.g., address, date of birth, etc.) is a strong evidence. It is important that a policy framework allows for both kinds of evidences
- **Ontologies:** as stated above, policies will be exchanged among entities within the lifelong learning infrastructure. Although the basic constructs may be defined in the policy language (e.g., rule structure and semantics), policies may be used in different applications and even define new concepts. Ontologies help to provide well-defined semantics for new concepts to be understood by different entities.

IV. USING POLICIES FOR LIFE-LONG LEARNING

As it has been previously presented, policies are a flexible means to describe how a system should act, depending on certain conditions and events. In this section we show that with all the described features a policy framework can serve as a behavior control for an open eLearning environment and as a flexible means for learners to personalize the eLearning system and tailor it to their specific needs. In our scenario, when Alice uses an open learning environment, she implicitly exploits policies (statements controlling the behavior of her system) in several ways. She makes use of restrictions for incoming chat connections or conditions under which Alice’s locally stored resources (either documents or her credit card number) may be disclosed, content providers specifying whether a resource is free-of-charge or at cost (and the payment methods together with business rules like e.g., discounts) or general statements indicating how some entity (e.g., a software agent) should react to a specific event. An open life-long learning infrastructure must provide sufficient functionalities in order to support all these situations. Aligned to the scenario provided in Section II, the rest of this section describes how policies and their features can be used in order to cater learners sufficient flexibility and adaptivity.

A. Personalizing the Flow of Communication

As described in our scenario, communication is an important part of learning and due to the web technology communication via chat, email-subscription, etc. it is supported by almost all eLearning applications. For example, in our scenario, Alice’s eLearning client offers a chat tool in order to induce the communication between her, her fellow students, and her tutors. However, such a chat tool might overload the user because of too many chat requests. Some of them may be spam, some may concern urgent

issues about Alice’s business trips, others may be requests from her fellow students, and so on. Policies offer a means to adapt the control of Alice’s chat client and therefore cope with the plethora of chat requests. Since policies are declarative Alice can easily define who (i.e., business contacts, fellow students, etc.) may send her a chat message at what time (i.e., leisure time, during business trip, etc.) as well as whom she considers as business contact or as friend.

At the same time, since the policy framework included in the eLearning client offers explanations, each requester who wants to chat with Alice during working time, receives an explanation about why it is currently not possible to chat with her. As we showed above, policies are also suitable for the filtering of emails: automatic replying or forwarding, removal of spam, etc. Therefore, policies ease the handling of the communication flow and take the burden of browsing through all the incoming messages away from the learner.

B. Authorization and Trust

Policies allow users and systems to characterize new users or systems by their properties and not simply by their identity (crucial in an open environment where complete strangers may interact with each other). This enables a new kind of access control in an *open* environment. For example, if Alice defines in a policy that any user of her community can access a certain set of the resources stored on her computer, this access right does not need to be adapted when the community changes or new people join.

When performing a purchase through her eLearning client, Alice can exploit the policy-driven negotiation functionality her eLearning client provides. Negotiations can be (semi)automatically performed among entities driven by their policies [15]. She may have protected her credit card with a policy. This policy states that she is willing to disclose it only to content providers, that are certified by the Better Business Bureau (BBB) and therefore considers such providers as trusted. The Better Business Bureau (BBB) seal program guarantees that its members will not misuse or redistribute disclosed information. Following Alice’s example, a negotiation is needed, because—in order to let Alice prove that she has a valid credit card that can be charged—the content provider has to prove first its certification by the Better Business Bureau. This negotiation is depicted in Figure 2.

In the previous example, if the content provider does not accept Alice’s credit card, she would get a detailed explanation helping her to know what to do next in order to successfully complete the buying process. A statement like “This credit card is invalid because it expired” would be indeed more useful than a simple deny as “Invalid credit card”. Aligned to our eLearning scenario, Alice might want to exploit negotiations in order to decide which fellow students are entitled to access the resources she created and stored on her computer, thereby, requiring other trust evidences from them, such as some digital society membership credential.

As shown, policies especially provide sophisticated methods to control who accesses resources. This is particularly helpful if a learner wants to share homeworks or lecture notes with fellow students. According to certain properties of the requester, access to this information may be granted or not. Therefore, authentication plays a remarkable role in life-long learning. Actions are a desired feature in this context: a fellow student who

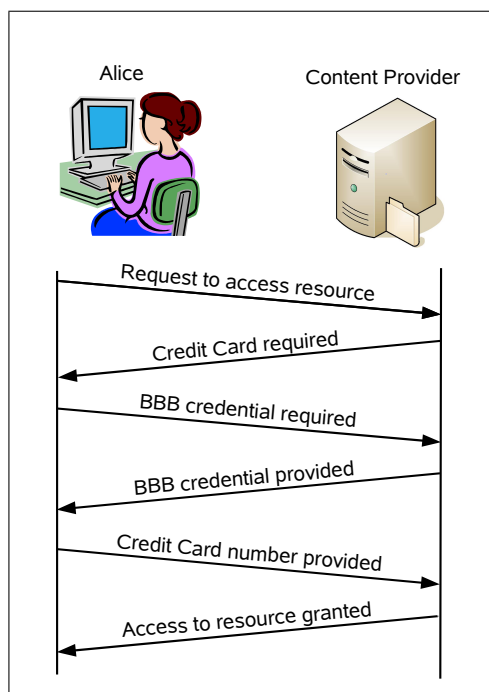


Fig. 2. Example negotiation sequence between Alice and the learning content provider

wants to access Alice's exercises may have to provide a digital membership certificate first proving that she enrolled to the same course. Sending this certificate could also be integrated into the specification of the policy via an action.

C. Motivating and Triggering of Learning

Policies can be used to control the behavior of software agents in order to directly react to a learner's behavior or learning progress. Agents can send automatic notifications via chat messages. They may also drive or define the rules of electronic games and simulations for educational purposes or many other approaches in order to increase a learner's motivation while learning. Due to the declarativity of policies, all these behaviors can be configured and driven via policies based on the learner's properties and behavior. The automatic chat message in our scenario is one example. Alice wants to receive such a message in case she was inactive for more than a week. The triggering can be done via a policy language enabling actions: the sending of a chat message itself is not part of the policy framework but it can still be called via an action construct in the policy language.

D. Personalizing the Content Presentation

Adaptivity of eLearning content has been the focus of research for some decades already (see [16]). By using policies, the adaptive behavior of the system, and in particular the adaptive presentation of learning content can be defined in a declarative way. In our scenario, Bill may specify that certain items in his eLearning course are shown or hidden, according to the results of a formative assessment of a learner or provided at run-time by the learner (therefore, even allowing for partial profiles stored at the user machine). If institutions agreed on profile models (e.g., competence descriptions) this could even lead to solutions for exchange of partial user profile information between different

systems the learner is using (either distributed profiles and/or partial profiles stored at the learner machine).

V. POLICY FRAMEWORKS AND THEIR FEATURES

In this section we first briefly introduce the most prominent policy languages. Then we provide a comparison of these languages in order to select one which provides the required and desired features described in the Section III. At the creation time of this paper, a variety of policy languages have been developed and are currently available. Out of those, we chose the most popular and widely used languages.

- **Ponder** [3] is a policy language that was meant to help local security policy specification and security management activities, therefore typical addressed application scenarios include registration of users or logging and audit events, whereas firewalls, operating systems and databases belong to the applications targeted by the language
- **WSPL** (Web Services Policy Language) [6] supports description and control of various aspects and features of a web service
- **KAoS** [1] addresses Web Services and general-purpose grid computing, although it was originally oriented to software agent applications where dynamic runtime policy changes need to be supported
- **Rei** [5] was primarily designed to support pervasive computing applications, Such applications are meant to be run on mobile devices which use wireless networking technologies to discover and access services and devices
- **PeerTrust** [2] is a simple yet powerful language for trust negotiation on the Semantic Web based on distributed query evaluation
- **Protune** [4] supports trust negotiation as well as a broad notion of "policy" and does not require shared knowledge besides evidences and a common vocabulary
- **XACML** (eXtensible Access Control Markup Language) [7] is an XML-based policy language and was meant to serve as a standard general purpose access control language ideally suitable to the needs of most authorization systems

In the following section we provide a comparison of these languages in terms of whether they are suited for eLearning scenarios.

A. Comparison of Policy Frameworks

The number and variety of policy languages proposed so far is justified by the different requirements they had to accomplish and the different use cases they were designed to support. In the following we compare the different features offered by the main policy languages in detail (see Table II for a summary).

- **Negotiations:** [6] adopts a broad notion of "negotiation", namely a negotiation is supposed to happen between two peers whenever (i) both peers are allowed to define a policy and (ii) both policies are taken into account when processing a request. According to this definition, WSPL would support negotiations as well. However, for the sophisticated negotiations we talk about in this paper, we need to adopt a narrower definition of "negotiation" by adding a third prerequisite stating that (iii) the evaluation of the request must be distributed, i.e., both peers must locally evaluate the request and either decide to terminate the negotiation or

send a partial result to the other peer who will go on with the evaluation. Whether the evaluation is local or distributed may be considered an implementation issue, as long as policies are freely disclosable. Under a conceptual point of view, distributed evaluation is required as soon as the need for keeping policies private arises: if policies were not be private, simply merging the peers' policies would reveal possible compatibilities between them. This is not the case in life-long learning, where policies may be sensitive. Imagine, Alice states that her friends can contact her via instant messenger in her leisure time but not any of her business contacts: most probably Alice does not want her business contacts to see this policy

- **Evaluation:** In languages that support negotiations, policy evaluation is distributed: at each step in the negotiation process, a peer sends the other one information which (possibly) lets the negotiation advance. However, each peer can terminate the negotiation without successful completion at any time. The evaluation is supposed to be performed locally by languages which do not support negotiations, although some of them may allow to split policies in several nodes and provide some means for collecting all fragments before (or during) the evaluation. Finally some languages like Ponder neither support distributed evaluation nor distributed policies [17]
- **Delegation** Delegation is often used in access control systems to cater for temporary transfer of access rights to agents acting on behalf of other ones (e.g., passing write rights to a printer spooler in order to print a file). The right of delegating is a right as well and as such can be delegated, too. Some languages provide a means for cascaded delegations up to a certain length (1 in Ponder, 2 in Rei): such languages provide a specific built-in construct in order to support delegation. Protune does not provide high-level constructs to deal with delegation but simulate them by exploiting more fine-grained features of the language: this has the remarkable side-effect of allowing to explicitly set the desired length of a delegation chain (as well as other properties of the delegation). Delegation (of authority) can be expressed in PeerTrust as well, whereas KAOs, WSPL and XACML do not support delegation

As shown in Table II, Ponder, Rei, PeerTrust and Protune support delegation but only PeerTrust and Protune also allow for negotiations and both strong and lightweight evidences. However, Protune is the only policy language also supporting advanced explanation mechanisms and seems to be one the most complete language available (like it has been pointed out in [18]). On the other hand, Protune assumes by default that resources are private, therefore not allowing for the specification of negative authorizations, which is a feature supported by other frameworks like Rei or KAOs. However, Protune does not only allow for distributed evaluation of policies (thereby allowing policies to be kept private), but also sufficient open source implementations are available, making this language easily accessible, usable and extendable. For this reasons, and since negative authorization policies are not necessarily needed in our application scenario, we decided to exploit Protune in our implementation. In the following a brief overview of the Protune language is provided as well as the description of its application to the scenario described in Section III.

B. Protune language

The previous section provided a description of the most prominent policy languages defined up to the date of creation of this paper. In this section, we introduce the Protune policy language, which seems to be the most suitable policy language for implementing our scenario.

It is important to note that even though we provide a detailed description of the PROTUNE language and its reasoning process, users will not be requested to specify their policies in a rule-based logic language such as Protune. In contrary, end-users will be able to select and instantiate existing policies from a standard library⁵ or, for advanced users and administrators, appropriate tools for the specification of new policies will be provided. In fact, most of the policy languages presented in section V provide management editors that help end-users and administrators to create and manage their policies.

The Protune policy language is based on regular logic program rules [19] of the following form

<pre> if L₁ and ... and L_n then A₁ and ... and A_m </pre>
--

which, according to the standard Logic Programming notation, can be written as

<pre> A₁ ← L₁, ..., L_n. ... A_m ← L₁, ..., L_n. </pre>
--

A_1, \dots, A_m represent standard logical atoms (called the *heads* of the rules) and L_1, \dots, L_n (the *bodies* of the rules) are literals, that is, L_i equals either B_i or *not* B_i , for some logical atom B_i . In the following we present a simple rule which Sam evaluates each morning before leaving home.

<pre> samReadyToLeaveHome ← samHaveKeys, samHaveWallet, samHaveMobilePhone, samHaveWatch, samHaveTransponder, samHaveTissue. </pre>

The intended meaning is as follows: Sam is ready to leave home if he has his keys, wallet, mobile phone, watch, transponder and a tissue. The main drawback of this rule is that it only applies to Sam and, in case also Tom needs to perform the same check before leaving home, a new rule has to be defined for him. Fortunately Logic Programming allows to parametrize atoms, so that they can refer to entities not known in advance, like in the following example.

<pre> readyToLeaveHome(X) ← haveKeys(X), haveWallet(X), haveMobilePhone(X), haveWatch(X), haveTransponder(X), haveTissue(X). </pre>

⁵E.g., similar to the mechanisms used in Microsoft Outlook for the instantiation of filtering rules.

	KAoS	Ponder	PeerTrust	Protune	Rei	WSPL	XACML
Authorization types	Positive & negative	Positive & negative	Positive	Positive	Positive & negative	Positive & negative	Positive & negative
Negotiations			supported	supported			
Evaluation	Centralized	Centralized	Distributed	Distributed	Centralized	Centralized	Centralized
Explanations				Advanced			Simple text
Kind of Evidences			Strong & lightweight	Strong & lightweight	Strong		
Delegation		supported	supported	supported	supported		
Ontology support	Actions, actors...	Roles, groups...		Actions	Classes, properties...		
Open source implementation			supported	supported			supported

TABLE II
DIFFERENT POLICY LANGUAGES AND THEIR FEATURES

In this example, X represents a variable (which may refer to whichever entity) and the intended meaning is as follows: some entity is ready to leave home if *the same entity* has its keys, wallet, mobile phone, watch, transponder and a tissue. If the variable X is unified with (i.e., replaced by) ' Sam ', the rule above is semantically equivalent to the aforementioned $samReadyToLeaveHome$.

Finally, notice that rules can be arbitrarily nested. For instance, the rule $readyToLeaveHome(X)$ states that an entity is ready to leave home if (among other things) it has its keys. This may in turn mean that the same entity must have the main entrance key, the back entrance key and the post box key, as it is stated in the following rule.

$haveKeys(X) \leftarrow$
 $haveMainEntranceKey(X),$
 $haveBackEntranceKey(X),$
 $havePostBoxKey(X).$

Rules can be reasoned over, i.e., rules can be exploited to check whether some statement holds (i.e., is true). During the reasoning process all applicable rules are checked till either one of them is successfully evaluated (thereby providing a proof that the original statement holds) or no more rules are applicable (thereby providing a proof that the original statement does not hold). In the following, we sketch the reasoning process for checking whether it holds that ' Sam ' is ready to leave home (i.e., whether the $goal\ readyToLeaveHome('Sam')$ holds).

Check whether $readyToLeaveHome('Sam')$ holds, i.e.,
Check whether $haveKeys('Sam')$ holds, i.e.,
Check whether $haveMainEntranceKey('Sam')$ holds.
Check whether $haveBackEntranceKey('Sam')$ holds.
Check whether $havePostBoxKey('Sam')$ holds.
Check whether $haveWallet('Sam')$ holds.
Check whether $haveMobilePhone('Sam')$ holds.
Check whether $haveWatch('Sam')$ holds.
Check whether $haveTransponder('Sam')$ holds.
Check whether $haveTissue('Sam')$ holds.

In addition to usual Logic Programming-based languages, Protune provides policy-oriented features like support to actions, evidences and metapredicates.

- **Actions:** Protune allows to specify actions within a policy: typical examples of actions are: sending evidences, accessing legacy systems (e.g., a database) or environmental properties (e.g., time). Actions are represented as usual predicates (called *provisional* predicates). Provisional predicates hold if they have been successfully executed

- **Evidences:** Protune allows to refer to strong and light-weight evidences (i.e., credentials and declarations) from within a policy. Evidences can be regarded as a set of property-value pairs associated to an identifier. Each property-value pair is represented according to an object oriented-like dot notation $id.property : value$
- **Metapredicates:** Protune allows to define properties of predicates, i.e., predicates about predicates. These predicates are called *metapredicates* and are associated with property-value pairs. They are represented through a notation close to the one used for evidences, namely $predicate \rightarrow property : value$. Rules containing metapredicates are called *metarules*. Metarules are typically exploited to assert some information about predicates occurring in a policy, e.g., the type of the predicate (property `type`, e.g., `provisional`) or some directives for the verbalization of the predicate which are meant to be used by the explanation facility (property `explanation`). Some properties apply only to provisional predicates: the value of the property `ontology` is the identifier of the action associated to the provisional predicate as reported in some ontology in order to achieve a common understanding of an action among several parties. The property `actor` (resp. `execution`) specifies which peer (either the requester or the provider) should carry out the action (resp. when the action should be performed)

In the remainder of this section, we exploit a policy fragment in order to introduce how Protune policies look like and which is the interplay between Protune rules and metarules. We revisit this policy fragment in Section V-C.

(1) $is_colleague(Name) \leftarrow$
 (1.1) $credential(C),$
 (1.2) $C.type : employee,$
 (1.3) $C.owner : Name,$
 (1.4) $C.issuer : 'SomeCompany'.$

(2) $is_colleague(_) \rightarrow type : abbreviation.$

(3) $credential(_) \rightarrow type : provisional.$

(4) $credential(_) \rightarrow actor : peer.$

This policy fragment contains a rule (1) and three metarules (from (2) to (4)). The rule states that the predicate $is_colleague$ holds if each literal in the body of the rule holds. The intuition behind that rule is that a credential is required from the communicating party. The property fields of this credential need to have

specific values. For example, the type of the credential needs to be *employee*, meaning that the credential states that the owner is member of a certain company, in our case '*SomeCompany*'. *Name* is a variable and should be the same name as stated in the rule head (1). This policy may for example serve Alice's learning client in order to find out if someone she communicates with is really working for her university or just pretending.

- Metarules (2) and (3) state that predicates *credential* but not *is_colleague* is a provisional predicate (i.e., represents an action)
- Metarule (4) states that the action associated to *credential* must be performed by the other peer. In the following we assume that provisional predicate *credential* is associated to the action of sending a credential to the other peer

Assuming that we want to check whether Bob is a colleague, the policy fragment will be evaluated against the goal *is_colleague('Bob')* as follows:

- line (1.1) checks whether a credential *cred* has been sent by the other peer. If it is the case the evaluation proceeds, otherwise a failure is reported
- the lines from (1.2) to (1.4) check whether the values of the properties of *cred* correspond to the ones listed in the body of the rule. If it is the case the evaluation proceeds, otherwise a failure is reported

C. Motivation Scenario (revisited)

So far, we provided the description of all the properties a policy language must provide in order to address our scenario. Further, we described the language we chose, i.e., Protune. In this section we model the scenario introduced in section II by formalizing Alice's policies in the Protune language. We further present how the introduction of these policies and their evaluation cater important benefits for Alice, namely explanations and negotiations.

In our running example, Alice needs to specify that during work time her chat facility is only allowed to accept incoming messages from business contacts and other employees of her company. Intuitively, this policy states that chats are allowed for business contacts as well as for colleagues at working time (first and second rule) and for all other people only at leisure time.

```

allow(chat(Requester, init_conversation)) ←
    working_time,
    is_business_contact(Requester).

allow(chat(Requester, init_conversation)) ←
    working_time,
    is_colleague(Requester).

allow(chat(Requester, init_conversation)) ←
    leisure_time.

allow(chat(Requester, Action)) → explanation :
    Requester & "can contact Alice for action " & Action.
    
```

Further, *working_time*, *leisure_time*, *is_business_contact* and *is_colleague* may be defined as

```

working_time ←
    time(T),
    T > 8 : 30,
    T < 17 : 00.

leisure_time ←
    not working_time.

is_business_contact(Name) ←
    retrieve_contact(Name),
    Name.category : 'Business'.

is_colleague(Name) ←
    credential(C),
    C.type : employee,
    C.owner : Name,
    C.issuer : companyXYZ.

working_time → explanation : "It is working time".
leisure_time → explanation : "It is leisure time".
time(T) → explanation : "Time is " & T.
...
    
```

Especially important in this example is that by using Protune, explanations are available. In both policy excerpts, the last rules are metarules describing how to explain the corresponding predicate (the symbol & refers to string concatenation).

If we focus on the aspect, how Protune supports explanations, we can have a look at the last rules in the previous policy. Those metarules describe, how to explain the corresponding predicate. For example: If Bob, who is a friend of Alice, tries to contact Alice during her working time, the following explanation will be automatically generated from the specified policy [13]:

```

It can't be proved that
Bob can contact Alice for action init_conversation because
there is no Requester such that
    Requester is a business contact           [details]
AND
there is no Requester such that
    Requester is a colleague                 [details]
AND
it is not true that
    it is leisure time.                       [details]
    
```

In this explanation, the statements which are true and do not depend on the requester are hidden. Hence, the explanation is focused on the conditions which are not fulfilled and not crowded with conditions which are (possibly trivially) true. However, full explanations providing both fulfilled and not fulfilled conditions can be generated on demand. In addition, clicking on [details] in a line provides a new explanation for the concept described in that line. For example clicking on the last [details] link would yield an explanation, what exactly is meant by leisure time, i.e., which time frame Alice considers leisure time.

Alice might also have the following policy protecting her credit card when trying to access on-line resources at different learning resource providers. Intuitively, this policy states that the credit card can be released to trusted parties and that a trusted party is a party providing a BBB credential.

```

allow(access(CC)) ←
  CC.type : credit_card,
  CC.owner : Name,
  trusted(Name).

trusted(User) ←
  bbbMember(User).

bbbMember(User) ←
  credential(C),
  C.issuer : 'Better Business Bureau',
  C.name : User.

```

The provider she is contacting has the following policy specifying the payment conditions.

```

allow(access(Course)) ←
  price(Course, Price),
  paid(User, Course, Price).

paid(User, Resource, Price) ←
  credential(CC),
  CC.type : credit_card,
  CC.owner : User,
  authenticated(User),
  charged(Resource, Price, CC).

allow(release_credential(bbbCredential)).

```

Let us assume that Alice finds a course she is interested in and requests access it. As soon as this request is notified to the content provider, a dynamic negotiation is initiated by the policies defined by Alice and the provider (depicted in Figure 2). This negotiation is intended to satisfy the policies of the communicating parties in an iterative way and enable on-line interaction.

VI. IMPLEMENTATION

This section briefly presents the PROTUNE policy framework as well as its main components. It also introduces an on-line demonstration of some of the features described in this paper applied to an eLearning scenario. Finally, it evaluates the performance of the PROTUNE framework.

A. PROTUNE Policy Framework

The PRovisional TrUst NEgotiation framework PROTUNE [4] aims at combining distributed trust management policies with provisional-style business rules and access-control related actions. PROTUNE's rule language extends two previous languages: PAPT [20], which until 2002 was one of the most complete policy languages for policy-driven negotiation, and PeerTrust [2], which supports distributed credentials and a more flexible policy protection mechanism. In addition, the framework features a powerful declarative meta-language for driving some critical negotiation decisions, and integrity constraints for monitoring negotiations and credential disclosure.

PROTUNE provides a framework with

- a trust management language supporting general provisional-style⁶ actions (possibly user-defined)
- an extensible declarative meta-language for driving decisions about request formulation, information disclosure, and distributed credential collection

⁶Authorizations involving actions and side effects are sometimes called provisional.

- a parametrized negotiation procedure, that gives a semantics to the meta-language and provably satisfies some desirable properties for all possible meta-policies
- integrity constraints for negotiation monitoring and disclosure control
- general, ontology-based techniques for importing and exporting meta-policies and for smoothly integrating language extensions
- advanced policy explanations⁷ in order to answer why, why-not, how-to and what-if queries [13]

The PROTUNE policy framework offers a high flexibility for specifying any kind of policy, integrates external systems at the policy level and provides facilities for increasing user awareness, like for example natural language explanations of the policies. It is entirely developed in Java, what permits its integration into web environments as an applet (without requiring the installation of any additional software). Figure 3 depicts the high level architecture of a PROTUNE Agent. It is composed by the following modules.

- *Communication Interface* It is in charge of the communication with other parties. Some examples of possible interfaces are secure socket connections or web services
- *Internal Java API* This API can be used by Java programs in order to integrate the policy framework's functionalities
- *Policy Engine Distributor* In case more than one policy engine exist, this component is in charge of forwarding any request to the appropriate one
- *Policy Engine* Specific policy engine in charge of processing requests. Currently a PROTUNE engine and a PEERTRUST engine are implemented
- *Credential Selection and Termination Algorithm* This is a pluggable component that specifies the general negotiation strategy of the agent (see previous sections for a more detailed description)
- *Inference Engine* It is in charge of checking whether the (local) policy has been fulfilled, as well as of other evaluation processes like extracting from the local policy (resp. from the policy of the other peer) the actions the current peer wants to execute (resp. the actions the other peer wants the current one to execute)
- *Execution Handler* Responsible for executing actions and package calls specified in the policies
- *Credential Repository* This package is in charge of loading the local credentials, providing them when required and checking that received credentials during a negotiation are not forged
- *RDBMS* This package is in charge of executing database queries to a relational database
- *File System* This package is in charge of executing queries based on regular expressions on specified files in a file system

B. On-line demonstration

As part of our investigation we have developed an on-line demonstration as a proof-of-concept of the feasibility of using policies for advanced eLearning scenarios. Since our goal was to show the interactions in which policies are involved, this

⁷A full demo is available at <http://cs.na.infn.it/reverse/demos/protune-x/demo-protune-x.html>.

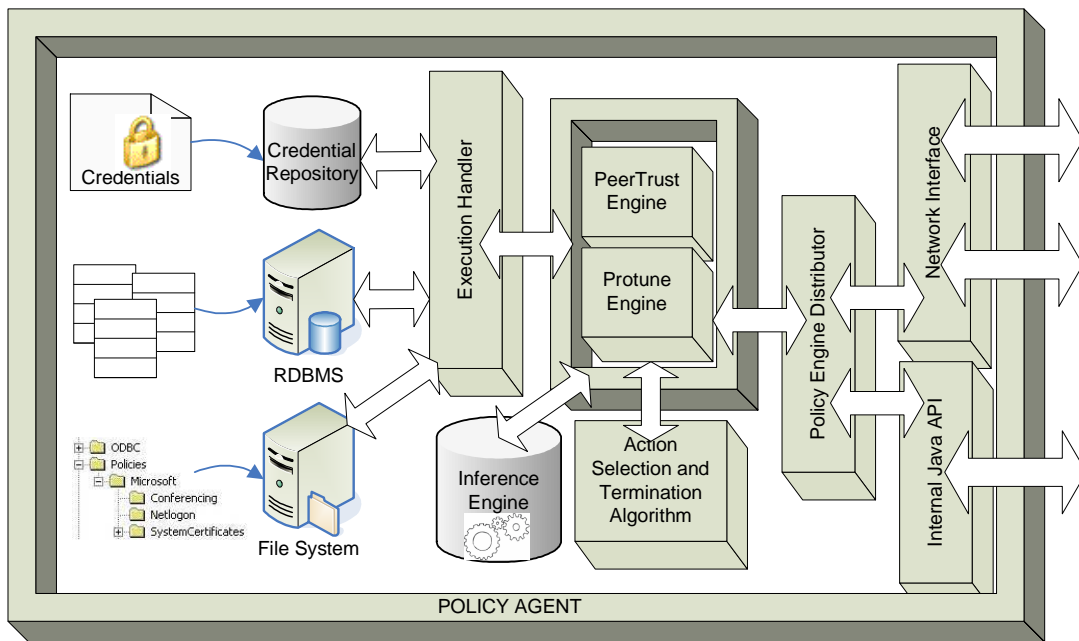


Fig. 3. Policy Framework Architecture

demonstration adds actual policy-based reasoning and interactions to a learning management system scenario. The demo⁸ is available at <http://policy.l3s.uni-hannover.de:9080/policyFramework/elearning/>.

It is important to note that the personalization and all the policy interactions are perfectly integrated in the demonstration and performed at runtime (and can even be modified by the user accessing the demonstration). On the other hand, the learning management system is just a mock-up and does not provide real functionality. For example, there is no actual instant messenger or file sharing components integrated in the demo, since it is out of the scope of this paper.

The demonstration site downloads an applet to the user's computer in order to provide the reasoning and negotiation capabilities required for the advanced interactions with the learning management system. Once loaded, the page shows the following elements (see also Figure 4)

- a personalized greeting message
- a list of the files shared by some colleagues
- an instant messenger with the available on-line contacts
- a lesson about Java
- a learning agent, which shows some messages to the learner
- a list of books related to the displayed lesson, which the learner may be interested in

Each component allows the user visiting the demonstration site to perform several interactions

- *greeting message*. Before the content is generated, the server requires the user to identify herself. Depending on the answer of the user, her name or a request to register will appear
- *file sharing*. When clicking on any of the files, a negotiation with the server (emulating a negotiation with the other learner) takes place in order to allow access to the file. Only learners holding and disclosing a student card

⁸Java 1.6 and Javascript enabled are required to run the demo.

from the learning management system can retrieve the files. Otherwise, an appropriate explanation is shown

- *instant messenger*. If there is a request for chatting with any of the on-line contacts, a request (possibly involving a negotiation) is performed in order to find out whether the other party is accepting chats at that very moment. Otherwise, an appropriate explanation is shown
- *main pane with lesson*. Before delivering the content for the Java course, the server requests the client to provide a certification of her advanced knowledge in the topic. If that certificate is provided, the server delivers a "Java Programming for experts" unit. Otherwise, it delivers the "Java Programming for beginners" unit
- *learning agent*. Depending on the identity of the learner, the agent will show different personalized messages oriented to helping the learner
- *book list*. Depending on the level of knowledge of the learner (e.g., whether the learner has provided a certification of her advanced knowledge of Java), the list will contain different books, either for Java beginners or experts. In addition, if the user clicks on any of them, a negotiation will be performed in order to check whether the learner has a subscription with the publisher or whether she buys the book by providing a valid credit card. Otherwise, an appropriate explanation is shown

The demo is initialized in a way that the learner automatically discloses for instance her identity ("Alice") and the certification for advanced Java. However, this behavior can be changed by opening the policy editor (a link is provided in the site) and commenting or uncommenting the available policies. Those changes will have as consequence that the interactions with the learning management system will, at run-time, generate different results.

In summary, the on-line demonstration shows that by using policies we enhance a static learning environment with flexibility, dynamicity, and interoperability: each user is allowed to adopt the

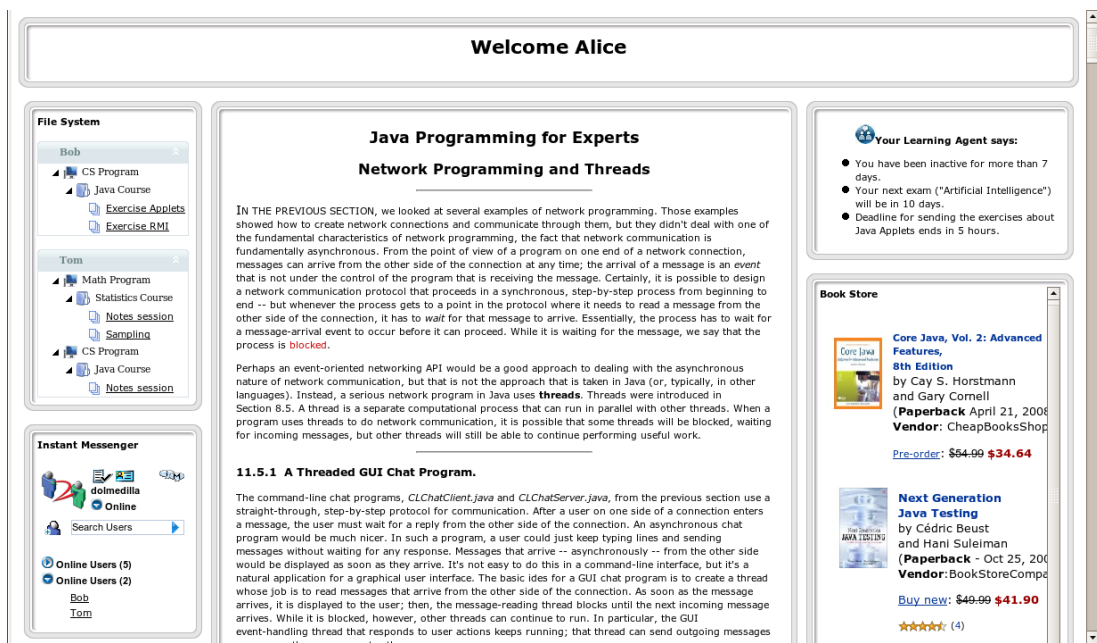


Fig. 4. Demonstration of the integration of a Policy Framework Architecture in an eLearning scenario.

behavior best suiting her needs and the adoption of the system can happen at runtime.

C. Evaluation of the Protune Architecture

For systems as described in this paper, the performance of the evaluation process of policies is a very important aspect. Even, if policies offer manifold possibilities, it is unlikely that users accept extraordinary long response times. The evaluation process is done by the policy engine, which is the heart of every policy integration. The policy engine operates on a specific set of policies to perform a logic reasoning process, resulting in an answer to the given query.

In order to evaluate the response time required by the policy engine and to observe the impact of its implementation into a productive environment, we assumed requests to a policy engine, running on a user's computer. Therefore, we used a Laptop using an Intel Core 2 Duo CPU T7300 with each 2.00GHz and 2,030 megabytes of RAM. The simulated user has a file containing the policies that will be loaded in the engine. We varied the number of policy rules in the file, assuming a minimum set of 100 policy rules, which already represents a very advanced and sophisticated behavior of the user's system driven by policies. Additionally, during the test we increased the amount of rules subsequently in steps of hundreds up to 800, an amount that we consider to be already an extraordinary big set of rules for a user but probably a regular amount for a server. Within the rule set, we also assumed a moderate dimension of complexity of the policy rules and also of connections between rules, that is, forcing the reasoning process to always have to evaluate several (possibly nested) rules. As depicted in Figure 5 the policy engine scales to the amount of rules in the policy file and the response time does not seem to increase.

The response time to one policy request is in average slightly above 200 milliseconds (on a non-optimized version of the PROTUNE engine with debugging enabled). Moreover, [21] investigates some possibilities to pre-evaluate policy-based decisions

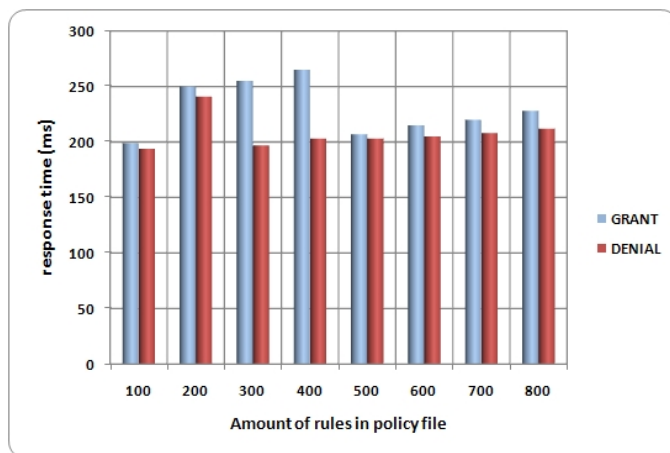


Fig. 5. Response times of the policy engine for different amounts of policy rules

in order to reduce response times in most cases to less than 10 milliseconds per request.

VII. RELATED WORK

To the best of our knowledge, using policy-based behavior control in technology-enhanced learning environments has not been extensively researched. An approach aiming at federated access control in web-service based repositories is presented in [22]. In order to allow for an appropriate access control, the policy language XACML and the federated trust framework Shibboleth have been extended and integrated into an ECL middleware. In this framework, policies are based on a simple attribute directory service.

In LionShare [10], a similar approach is used exploiting Shibboleth. Security is provided by so-called Access Control Lists expressed in XACML. These lists define which user can access which file depending on the user's properties, such as

the membership of a certain faculty. However, none of these approaches allows for expressive access control supporting e.g., action executions, negotiations or explanations and therefore, they do not meet the requirements identified in our scenario. Furthermore, using Shibboleth implies the existence of institutions users belong to - which is an assumption that does not apply in an open scenario for life-long learning.

[23] provides an abstract overview on privacy and security issues in advanced learning technologies, suggesting that policies suit well security purposes in an educational context. Besides the fact that in our work policies are applied more generally and not only to security, the work in [23] provides neither scenarios nor specific details about the usage of policies.

[24] deals with policies based on the Ponder policy language within the scope of collaborative eLearning systems. The use of policies in such a framework is basically restricted to role-based access control and therefore, does not match the needs of an open learning environment as described above.

The PRIME project [25] aims at developing a prototype of a privacy-enhancing identity management system. The project includes some scenarios demonstrating the applicability of the developed prototype where, among others, a learning environment setting is addressed. PRIME makes use of several existing languages and protocols like e.g. EPAL, XrML, P3P and the policy language XACML. The main focus of its learning scenario is obscuring the identity of the participating members in their different roles. However, the PRIME project obviously does not cover functionality provided by using a policy-based system as we suggest it in this paper.

Another eLearning project worth to be mentioned here is SeLeNe [26]. SeLeNe finished in 2004 and dealt with metadata of learning objects. One part of the project is highly related to this paper and covers the reactivity of learning object metadata [27]. One example application is the automatic notification on modifications of learning objects and the registration of a new user in a network showing an interesting (i.e., matching) profile. These reactivity features offer a complex change detection mechanism and base on so-called Event-Condition-Action Rules (ECA-Rules) defined on RDF. An ECA-Rule is a special kind of policy. SeLeNe discusses the basis of some of the features we require in our scenario (such as automatic notification via chat). However, the usage of policies we suggest in this paper allows for more complex conditions (not only based on RDF query languages) and actions (not only notifications) since we provide (among others) negotiations and arbitrary actions which are not part of the SeLeNe change detection framework.

Moreover, none of the approaches described above provides user awareness capabilities by, for instance, generating natural language explanations that may help the learners to understand the policies and decisions.

VIII. CONCLUSIONS AND FURTHER WORK

Open life-long learning environments require flexible and interoperable approaches which are easy to use and to personalize by learners and tutors. This paper described a scenario with advanced interactions among learners, tutors, agents and the learning management system. It also showed how policies can address the requirements extracted from such a scenario and provide benefits not only in flexibility and dynamicity but also additional features like reasoning and interoperability. The paper

gave an overview of existing policy frameworks and compared them according to a list of requirements previously identified. One of the policy languages was selected and used in order to specify policies that may be used at runtime to e.g., control access to resources, perform negotiations or generate explanations. Finally, we integrated the policy framework into a web-based on-line demonstration as a proof-of-concept of the feasibility of our approach, demonstrating the concepts described throughout the paper, and evaluated the performance of the policy evaluation process in terms of response times.

In our future work, we will investigate the integration of policies into existing eLearning systems (see [28]) such as Moodle [29] or ILIAS [30], but also into Adaptive Educational Hypermedia Systems such as AHA! [31]. We also plan to create more tooling support for the management of policies by non-expert users. Furthermore, PROTUNE does not currently support reactivity (event-condition-action rules) and we plan to explore such extensions to the language and framework in order to increase the number of possible scenarios it can address.

ACKNOWLEDGMENT

The authors' efforts were (partly) funded by the European Commission in the TENCompetence project (IST-2004-02787) (<http://www.tencompetence.org>).

REFERENCES

- [1] A. Uszok, J. Bradshaw, R. Jeffers, N. Suri, P. Hayes, M. Breedy, L. Bunch, M. Johnson, S. Kulkarni, and J. Lott, "Kaos policy and domain services: Toward a description-logic approach to policy representation, deconfliction, and enforcement," *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2003)*, p. 93, 2003.
- [2] R. Gavriloiu, W. Nejdl, D. Olmedilla, K. E. Seamons, and M. Winslett, "No registration needed: How to use declarative policies and negotiation to access sensitive resources on the semantic web," in *1st European Semantic Web Symposium (ESWS 2004)*, ser. Lecture Notes in Computer Science, vol. 3053. Heraklion, Crete, Greece: Springer, May 2004, pp. 342–356.
- [3] N. Damianou, N. Dulay, E. Lupu, and M. Sloman, "The ponder policy specification language," in *POLICY 2001: Proceedings of the International Workshop on Policies for Distributed Systems and Networks*. London, UK: Springer-Verlag, 2001, pp. 18–38.
- [4] P. Bonatti and D. Olmedilla, "Driving and monitoring provisional trust negotiation with metapolicies," in *POLICY 2005: Proceedings of the Sixth IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 14–23.
- [5] L. Kagal, T. W. Finin, and A. Joshi, "A policy language for a pervasive computing environment," in *4th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*. Lake Como, Italy: IEEE Computer Society, Jun. 2003, pp. 63–.
- [6] A. H. Anderson, "An introduction to the web services policy language (wsp)," in *POLICY'04: Proceedings of the Fifth IEEE International Workshop on Policies for Distributed Systems and Networks*. Washington, DC, USA: IEEE Computer Society, 2004, p. 189.
- [7] "OASIS eXtensible Access Control Markup Language," <http://www.oasis-open.org/specs/index.php/#xacmlv2.0>.
- [8] J. L. D. Coi, P. Kärger, A. W. Koesling, and D. Olmedilla, "Exploiting policies in an open infrastructure for lifelong learning," in *2nd European Conference on Technology Enhanced Learning (EC-TEL)*, ser. Lecture Notes in Computer Science, vol. 4753. Crete, Greece: Springer, Sep 2007, pp. 26–40.
- [9] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmer, and T. Risch, "Edutella: A p2p networking infrastructure based on rdf," 2001. [Online]. Available: citeseer.ist.psu.edu/boris01edutella.html
- [10] "The lionshare project," <http://lionshare.its.psu.edu/>.

- [11] T. Nabeth, A. A. Angehrn, P. K. Mittal, and C. Roda, "Using artificial agents to stimulate participation in virtual communities," in *CELDA*, Kinshuk, D. G. Sampson, and P. T. Isaacs, Eds. IADIS, 2005, pp. 391–394. [Online]. Available: <http://dblp.uni-trier.de/db/conf/iadis/celda2005.html#NabethAMR05>
- [12] F. Abel, E. Herder, P. Kärger, D. Olmedilla, and W. Siberski, "Exploiting preference queries for searching learning resources," in *2nd European Conference on Technology Enhanced Learning (EC-TEL)*, ser. Lecture Notes in Computer Science, vol. 4753. Crete, Greece: Springer, Sep 2007, pp. 143–157.
- [13] P. A. Bonatti, D. Olmedilla, and J. Peer, "Advanced policy explanations on the web," in *17th European Conference on Artificial Intelligence (ECAI 2006)*. Riva del Garda, Italy: IOS Press, Aug-Sep 2006, pp. 200–204.
- [14] P. Bonatti and P. Samarati, "Regulating service access and information release on the web," in *CCS '00: Proceedings of the 7th ACM conference on computer and communications security*. ACM Press, 2000, pp. 134–143.
- [15] W. Winsborough, K. Seamons, and V. Jones, "Automated trust negotiation," DARPA, Tech. Rep. TR-2000-05, 24 2000. [Online]. Available: citeseer.ist.psu.edu/article/winsborough00automated.html
- [16] J. S. Brown and P. Duguid, "Adaptive and intelligent web-based educational systems," *International Journal of Artificial Intelligence in Education*, vol. 13, pp. 156–169, 2003.
- [17] T. Yu, M. Winslett, and K. E. Seamons, "Interoperable strategies in automated trust negotiation," in *CCS '01: Proceedings of the 8th ACM conference on Computer and Communications Security*. New York, NY, USA: ACM, 2001, pp. 146–155.
- [18] C. Duma, A. Herzog, and N. Shahmehri, "Privacy in the semantic web: What policy languages have to offer," *8th IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY 2007)*, pp. 109–118, 2007.
- [19] J. W. Lloyd, *Foundations of logic programming*. New York, NY, USA: Springer-Verlag New York, Inc., 1984.
- [20] P. A. Bonatti and P. Samarati, "Regulating service access and information release on the web," in *ACM Conference on Computer and Communications Security*, 2000, pp. 134–143.
- [21] J. L. D. Coi, E. Ioannou, A. Koesling, and D. Olmedilla, "Access control for sharing semantic data across desktops," in *1st International Workshop on Privacy Enforcement and Accountability with Semantics (PEAS)*, Busan, Korea, Nov. 2007.
- [22] M. Hatala, T. M. Eap, and A. Shah, "Unlocking repositories: Federated security solution for attribute and policy based access to repositories via web services," in *ARES*. IEEE Computer Society, 2006, pp. 895–903.
- [23] K. El-Khatib, L. Korba, Y. Xu, and G. Yee, "Privacy and security in e-learning," *International Journal of Distance Education*, vol. 1, no. 4, 2003.
- [24] Yang, Lin, and Lin, "Policy-based privacy and security management for collaborative e-education systems," in *5th IASTED Multi-Conference Computers and Advanced Technology in Education*, Cancun, Mexico, 2002.
- [25] "PRIME: privacy and identity management for europe," <https://www.prime-project.eu>.
- [26] "SeLeNe: self elearning networks," <https://www.prime-project.eu>.
- [27] G. Papamarkos, A. Pouloussilis, and P. T. Wood, "Eca rule languages for active self e-learning networks," 2003, seLeNe Project Deliverable 4.4.
- [28] A. W. Koesling, E. Herder, and D. Krause, "Flexible adaptivity in aehs using policies," in *Conf. on Adaptive Hypermedia and Adaptive Web-Based Systems*, Hanover, Germany, Jul. 2008.
- [29] "Moodle: moodle open source elearning system," <http://moodle.org/>.
- [30] "ILIAS: ilias open source elearning system," <http://www.ilias.de/>.
- [31] "AHA!: adaptive hypermedia for all," <http://aha.win.tue.nl/>.



Juri L. De Coi is a Ph.D. student at the University of Hannover's Computer Science Department and a research scientist at the L3S Research Center. He received his master degree in computer engineering from Alma Mater Studiorum – Università di Bologna, Bologna, Italy.



Philipp Kärger is a research scientist at the L3S Research Center. He is doing his PhD in Computer Science at the Leibniz University Hannover. In 2006 he received his master degree in computer science from Saarland University, Saarbrücken, Germany.



Arne W. Koesling has graduated in computer science at the Carl-von-Ossietzky University of Oldenburg in Germany. He was responsible software engineer for a company for mobile services before he worked as independent consultant. He started as research scientist for the Hannover Medical School and the L3S Research Center in 2005.



Daniel Olmedilla is a project leader at L3S Research Center and the University of Hannover since 2005. Before joining L3S as researcher in 2002, he worked as consultant and project manager in IT companies. He received his master degree and Ph.D. in computer science from Universidad Autónoma de Madrid, Spain.