Interactive Protocol Visualization (and a WSPL Case Study)

Sean Cannella
(scannell@cs.brown.edu)

April 5, 2005

1 Abstract

While there is a strong body of research on algorithm animation, there is very little existing work on interactive protocols. What is described in this paper is a flexible and modular platform for visualizing interactive protocols. While the design will be layered to support additional protocols, the specific implementation goal is a case study: to create a viable visualization of a policy-based negotiation protocol. Sun's WSPL is the target protocol (even though it is not currently interactive), and we present a prototype of a visualization application for it as an example of the above framework.

2 Introduction

Security in modern information systems has become a major concern, given the ubiquity of such systems in multiple, almost fundamental, aspects of life. An often neglected component of security is visualization — the reality is that even a "secure" system is only truly secure if it used properly, and that requires that administrators and often even end-users understand the system and act accordingly. Usability studies on protocols like PGP [13] point out the fallacy of simply assuming that if a system can be used correctly that it will indeed be used correctly. For further background, as well as a case study in the domain of information integrity, see [2].

Thus, we turn our attention to interactive protocols that require some sort of negotiation component. There are many examples of such protocols, like automated trust negotiation [15] and Sun's Web Services Policy Language [1]. SSH can be viewed as a simple negotiation-based protocol, and many future protocols will be of this nature, most notably those coming out of interactive agent research in AI.

There are many aspects of "security" in these cases, where some sort of "agent" (whether it be a policy or a program) is acting on the user's behalf. Beyond authentication and integrity, there is the issue of ensuring that one's agent is acting according to one's wishes, and that adversarial agents are not managing to cheat the user — this is the issue we intend to address.

2.1 Road Map

First, Section 3 will outline policy languages and protocols and related visualization needs. Section 4 will discuss previous and related work. In Section 5, we will provide an overview of the architecture of the framework and present several potential use cases. Sections 6 and 7 will go into more detail about the data model and the specific visualization tools, and Section 8 will detail what has been implemented so far.

3 Policy Protocols and Languages

This section describes various policy protocols and languages where visualization is an important component in achieving understanding.

3.1 Automated Trust Negotiation (ATN)

A growing body of work (e.g. [15] [17]) is attempting to address the following scenario. Imagine a member of a health insurance plan wants to go to a prescription website and receive the appropriate discount on the purchase of a drug. The current process involves signing up for an account, adding another password to one's repertoire, and worst of all, divulging a lot of irrelevant private information that such sites generally require to receive such a discount (i.e. SSN.) Automated Trust Negotiation aims to solve this problem, by designing negotiation protocols that operate on signed credentials that allow one to prove information like possession of insurance without going through an account-creation procedure.

The best implementation work in the literature in this area comes from [6], and there is very little visualization work to assist in understanding these complex protocols. Winsborough and Li [14] illustrate that in current strategies for ATN, there are still potential holes with different kinds of inferences that can sometimes be made by an adversarial party, and thus this sort of visualization is critical in trying to understand where such vulnerabilities might arise. The overall architecture described in this paper aims to address protocols with this sort of complexity.

3.2 eXtensible Access Control Markup Language (XACML)

Sun's eXtensible Access Control Markup Language is an XML-based language that aims to simplify the use of access control policies in a variety of different domains. What follows is a coarse review of basic functionality — for more detailed information, see [10].

The main construct is the XACML `Policy`, which can be composed into `PolicySets`. Each `Policy` contains `Target`s to which it is relevant, and `Rule`s which govern access to a particular `Target`. The `Rule`s operate using `Attribute`s and `Function`s, which allow the user a generic way to specify constraints on accessing a particular service or resource.

Then, when an access check is performed, a `RuleCombiningAlgorithm`, or combiner for short, attempts to merge both parties' policies which have relevant `Target`s, by attempting to find agreeable service parameters that are legal within the `Rule`s of all relevant policies.

An implementation from Sun in Java is available, which takes care of most of the hard work and XML parsing. The review above is a gross-understatement of the its flexibility and power, which leads to an important observation. Because it is so generic and can be tailored for use in many different contexts, it is essential that a user understand how it is used for her specific application. This is the main reason for deciding to choose a XACML-based protocol as a target for study.

3.3 Web Services Policy Language (WSPL)

Sun's Web Services Policy Language [1] is a protocol based on XACML [10] that attempts to provide automated "negotiation" of service parameters for a given web service.

Paraphrasing [1], XACML is applicable to this problem insofar as a web service can be expressed as a XACML `Target`, and the desired aspects (or parameters) of the service can be expressed as XACML `Rules`. The `RuleCombiningAlgorithm` (combiner) here is a simple one: the goal is to make sure that one can reconcile the parameters desired by both the client and the provider of the service. The output of such a combination is a XACML `Policy` (or `PolicySet` which gives a set of values for the parameters that are agreeable to both parties, or nothing if there is no such set. As WSPL is designed today, this is very server-centric: the combiner simply chooses the best parameter for the server that the client is willing to tolerate (e.g. high cost, low speed.)

Yao [16] has implemented the combiner as described above according to [1]. However, this implementation does not include any visualization of what is happening, and that is precisely what we have implemented. It is worth noting that while WSPL does not have an interactive component, one can easily view any non-interactive protocol as a one-step negotiation protocol, so it is still relevant to examine this sort of protocol from the interactive viewpoint.

4 Previous Work

Visualization of procedures and protocols is not a new topic in general — there exists a rich body of work on algorithm animation for basic data structures, common algorithms of introductory computer science, programs [12] and large datasets [3]. However, most of this is limited to static protocols — the most complicated detailed visualizations are on protocols like TCP [18] [5]. Other work includes detailed visualization of packet paths for information technology purposes [19], but even this sort of work is limited to a relatively set execution, and there is only one active party and thus no negotiation. This is not to say that these works are not valuable — they simply do not adequately address algorithms and protocols with a strong game-theoretic component.

The only attempt at covering more interactive protocols is a paper by Papakosta and Burger [9], which introduces a language and model for describing interactive protocol visualization, including the token-ring protocol and two-way authentication. However, the paper is a self-proclaimed "position paper", and focuses more on modeling than on visual presentation. In addition, very little seems to have been done in this field since this paper. Thus, we attempt to address the dearth of research in this area, and supplement the work by Michael Shin [11] and Tracy Hadden [4] in the domains of ATN and XACML (WSPL) respectively.

5 Architecture Overview

This section describes the ideal layout for a general interactive protocol visualization framework. The actual implementation will be detailed in Section 8.

From an application perspective, the code should be broken up into two layers — a core layer, and a protocol-specific layer. The core layer should be comprised of the core UI code as well as functionality that is independent of the specific interactive protocol being visualized. Examples of such protocols are ATN and WSPL as described in Section 3. However, this divide would allow for extensions to support other protocols (and visualizations.)

From a UI perspective, the application is composed of "interactors." The term indicates different ways of visualizing **and** working with data. This is to somewhat modularize functionality, as well as allow users with specific interests, levels of expertise, and tasks to perform to gain access to powerful interactions without cluttering or confusing them with irrelevant or inappropriately detailed information or interfaces. This division is nothing new — it can be seen in many IDEs, including Eclipse, Oracle's JDeveloper and Microsoft's Visual Studio.

It is suggested that the application be implemented in some portable, flexible and powerful language (Java or C#), at the price of speed which is not so important in this domain. As far as support libraries go, Grappa [8] provides a library for graph-related visualizations.

Note that one of the design goals is to allow the user to provide suggestions to an agent about behavior and strategy. However, since no such APIs exist yet, one would either need to create some, or simply decide on an output format that would allow this to occur in the future.

Lastly, a necessary component of the visualizer is to have the file format for the user's "state", comprised of all known resources, policies, history, and meta-data. Utilizing such a file format should allow other developers to write their own interactor to suit their own uses and better handle new protocols that were not envisioned at the time of this authoring. The issue of encapsulating state is somewhat addressed in [9], but the treatment there focuses more on reproducing visualizations for learning or teaching environments, as opposed to analyzing real data and traces.

5.1 Use Cases

This subsection outlines a use case for three users of varying skill levels to illustrate the appropriateness of the proposed visualizations for ATN. The specific details of the interactors are discusses in Section 7.

1. Novice: The novice user is considering adapting ATN, but is suspicious that it may not protect his SSN properly. The user first goes to the Creation Interactor, and uses a plug-in to load a default database of resources and policies (that would be available as a complement, like a root certificate set.) Then, the novice goes to the Execution Interactor, and downloads a set of preexisting negotiations that correspond to the default database. (This would also be available — since one can choose what to save, it is reasonable that someone could share negotiations involving her SSN without actually revealing the number itself, and simply showing the related activity to her SSN resource.) The novice then chooses a strategy, and can execute against

default opponents, including a set of those designed to try to steal information from the user. Satisfied, the user decides to use ATN.

2. Intermediate: The intermediate user has likely already done the above, but is suspicious that it hasn't actually been working. The user runs the Static Analysis Interactor, and finds that that there is a potential flaw in one of her policies, say policy Z, due to a problem with a parent policy. The interactor suggests tightening the parent policy, and offers to do it automatically. The user does so, and then verifies the behavior as the novice user in the Execution Interactor.

3. Advanced: The advanced user is preemptively searching for security holes, and goes to the Execution Interactor to start replaying all transactions related to her SSN. She discovers that a potential choice for an agent was to divulge the SSN when it shouldn't have, and she manually goes to the Item Interactor to edit the policy.

6 Data Model

Building on top of XACML (or any policy representation), one of the prerequisites to construct the visualizer out of modular, related components is some sort of shared file format. This is a serious issue — tracing is a key component in many different domains, from this sort of policy visualization to software debugging.

State encapsulation is thus a necessity. This includes all policies, resources, and negotiations in which the user has participated. Using this format, one essentially keeps a log of one's view of the "universe" with respect to ATN objects, and allows first for each interactor to use different sections of data, and second allows for custom interactors to be written in the future to extend the current set of functionality. This will (ideally) be further explored in future work.

This raises the issue of synchronization. Making a change in one interactor potentially influences another. For example though, using a static checker to modify the properties of a resource or change a policy should be reflected in any view that involves that policy or resource.

In the framework described here, all interactors should be working off a single copy of the data — unless this proves to extremely cumbersome (which it has not in the prototype implemented), modifying information in one interactor should cause a global refresh in which all "interactors" update. Ideally, since one is encapsulating state at all times, an "undo" is very valuable here, as one can experiment with modifications easily and examine changes from different angles in different interactors.

7 Interactors

This section describes a base set of interactors necessary for the interactive protocol visualization framework we describe. The descriptions are relevant to ATN, but the images come from the prototype implementation for WSPL. The Creation Interactor image comes courtesy of Tracy Hadden and her tool for authoring of XACML policies.

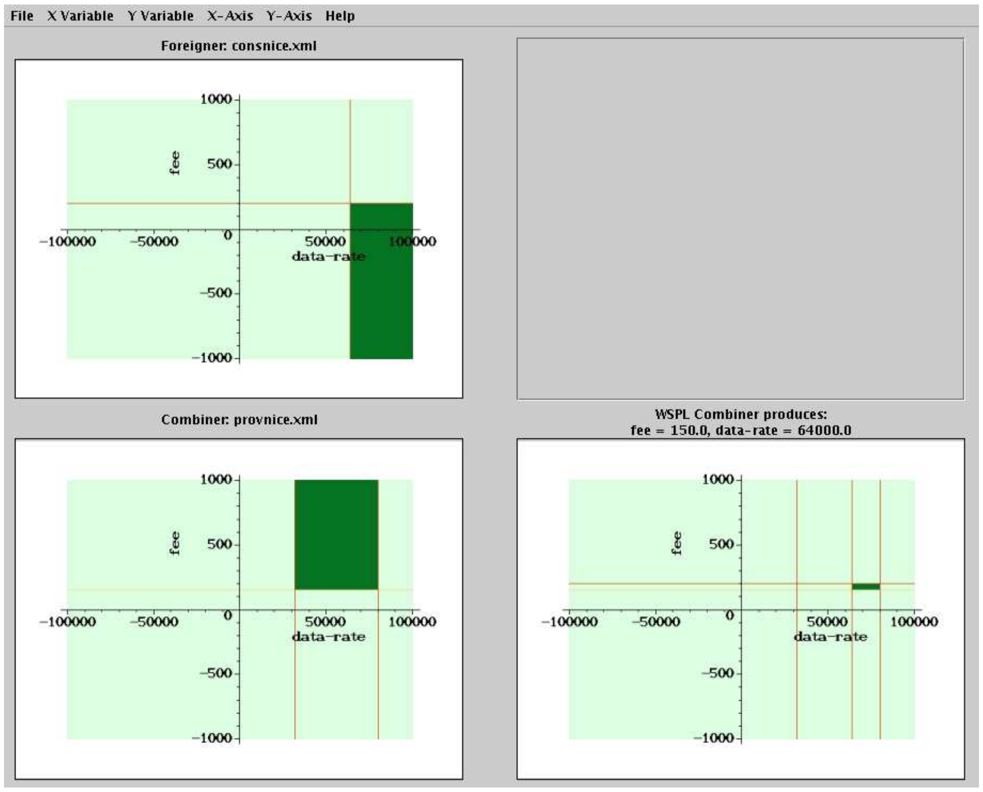


Figure 1: A sample image from an Execution Interactor for WSPL. The main components of this visualization are the inequality graphs representing the space dictated by the constraints of each policy at the left, and the merged graph for both policies at the bottom right. The full array of features is described in Section 8.2.

- Execution Interactor (Figure 1):

  The interesting core of the program, this interactor will allow the user to watch the execution of a negotiation. This is supported in two different modes: First, the user can watch the step-by-step execution of a recorded negotiation, to gain insight into the interactions of an agent. Second, the user can interactively make selections to explore strategies for ATN either as both parties in the negotiation, or as a single party an interact with an agent. Also supported is the ability to have two agents interact — this is ideal for a demo mode, or to test the functionality of a specific agent before a user entrusts it with guarding her data. Since we are not aware of APIs at the moment for interacting with agents, some rudimentary mechanism could be used to mimic interaction with an agent.

  The last mode mentioned is an appeal to the novice user; watching two agents interact step-by-step should help the user gain insight into the protocol. On the other hand, it (as well as the replay capability) appeals to the advanced user, since it allows one to test new agents as well as replay old interactions looking for potential bugs or security holes by viewing the net results of the exchanges.

  Visually, this will be primarily graph-based. Basic graph metaphors, like colors for different types of nodes (relevant meta-data, target in a graph, current choices, etc.) will be utilized. Further meta-data should be available in mouse-over annotations, and most importantly, in step-by-step mode, potential children should be displayed to help the user understand what is going on, rather than having them blindly input a policy choice. Without actual agents, legal moves will have to be otherwise determined to be visualized. A ranking system could assign colors to different choices based on suggestion from some entity, or rather, in simulation of a specific agent, the agent's choice could be highlighted, with the user able to override the choice with another legal move.

- Static Analysis Interactor (Figure 2):

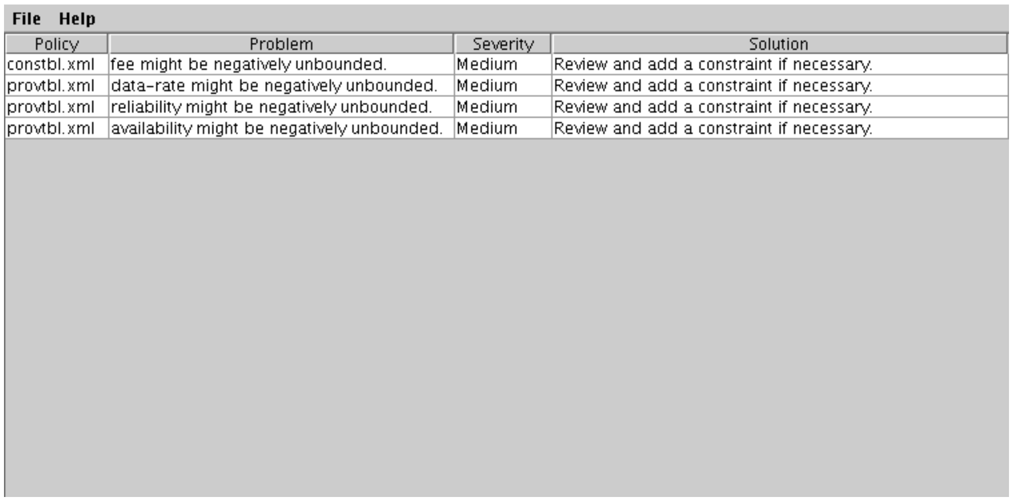
  The idea here is to create a view that allows both basic and advanced users to use some built-in static analysis tools as well as external modules to examine the current "state" on the user's system, comprised of all recorded executions of the protocol, as well as all known resources and policies. The purpose of this would be to look for common problems and potential flaws in the "state" and alert the user of potential privacy/security issues as soon as possible.

  Visually, this isn't a very graphical component — the plug-ins themselves would have control over what to display, since it would greatly depend on the analysis being performed.

  This could easily be extended to plug-ins which are much smarter, and perform deeper and more useful analysis.

- Item Interactor (Figure 3):

  The purpose of this interactor is to allow the user to visualize her working set of both resources and policies. In a sense, it is designed to provide a snapshot of the user's knowledge, without displaying specific interactions.



Policy	Problem	Severity	Solution
constbl.xml	fee might be negatively unbounded.	Medium	Review and add a constraint if necessary.
provtbl.xml	data-rate might be negatively unbounded.	Medium	Review and add a constraint if necessary.
provtbl.xml	reliability might be negatively unbounded.	Medium	Review and add a constraint if necessary.
provtbl.xml	availability might be negatively unbounded.	Medium	Review and add a constraint if necessary.

Figure 2: A sample image from a Static Analysis Interactor for WSPL. One can see that many constraints in these policies can potentially go negative, which does not make sense for attributes such as data-rate and fee.

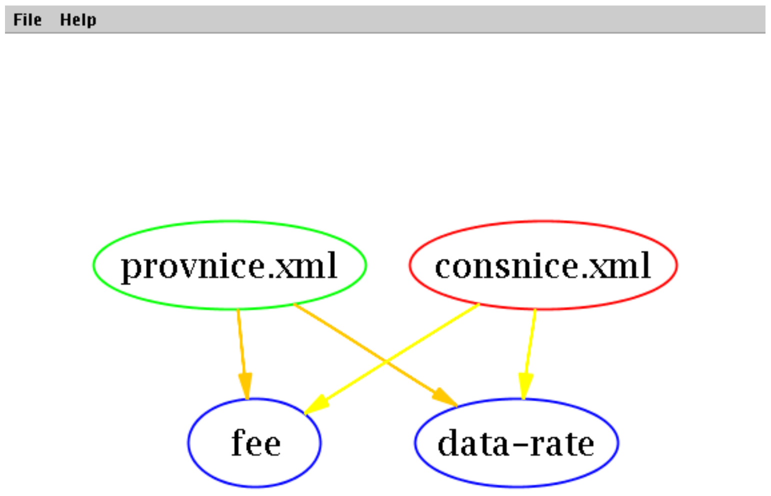


Figure 3: A sample image from an Item Interactor for WSPL. One can see from this figure that both policies affect the fee and data-rate attributes, and each policy's attributes is marked with edges of a different color.

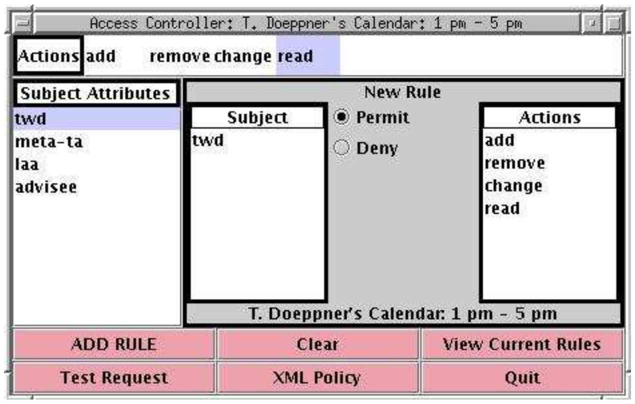


Figure 4: A sample image from a Creation Interactor for XACML.

This interactor is useful to users of all experience levels — for the basic user, this will provide an easy to access snapshot of the system with some consolidated details, without going into the details of specific interactions. Indeed, this is the core of the program for inexperienced users. Expert users will find this to be more of a summary interactor and likely find the Execution Interactor to be more effective in searching for problems.

Visually, the basic mode of this interactor is a three-window format, with two windows horizontal on top, and information about the selection in the window below. (Note that Figure 3 does not reflect this, due to the fact that WSPL policies are simple enough that the third window reflecting the information about all of the policies and resources suffices.) Policies will be displayed on the left, and resources on the right. Selecting one will highlight related elements in both lists, and dependencies and text will be rendered below. (Examples are a policy that refers to certain resources, and a credential resource that is related to other resources.) The text will contain any associated meta-data, as well as usage information that is obtained from the known executions of the system, and allow for potential threats to be examined.

A more advanced, but less important mode, will allow user-specified grouping of related resources and policies. This meta-data will allow a user to see amalgamated information, which may provide more insight than simple itemized information. This visualization is natural insofar as XACML provides support for relating policies and resources. Graph-based visualization is a natural method for this, since parent/child relationships are easily expressible in XACML.

- Creation Interactor (Figure 4):

  This component allows the user to introduce new policies and resources into the system. (Note that resource refers both to user-possessed resources, and resources that the user knows about — the user reasonably would want to refer to resources that she does not own but wishes to request of other parties in her policies.) Especially in a bootstrap scenario, this will be especially necessary, as the user has to have some way to initially configure her protocol-related universe before an agent can act on her behalf. It will also be designed to allow an external plug-in or program to interface, so that some more intelligent agent program could assist the user in intelligent additions of policies, or a plug-in which adds a large database of known resources with intelligent default meta-data to the system.

Novice users will likely make use of a plug-in that configures most of this for them, especially if there are available repositories of "good" policies and common resources. This interactor would be most useful to either the early adopter of ATN who must do this by hand, or knowledgeable users who understand enough and would need the ability to start adding policies to the system manually.

Like the Static Analysis Interactor, this component will also be light on the visual side, as it is only intended to allow users to add to the system — the actual visualization of the items added here is handled by the Item Interactor.

8 WSPL: A Case Study

This section details our prototype implementation of an interactive protocol visualizer for Sun's WSPL using Java 1.4.2.

Sun's public XACML implementation [10] is used to implement policies and resources, since all of the infrastructure is already there. Danfeng Yao's WSPL implementation [16] is used to actually get results for policy combinations that are visualized by the tool.

For the graph-based visualization in the Item Interactor, the library of choice is Grappa [8]. Grappa does not provide any layout, but allows one to use an external layout program to perform layout. However, **dot**, which is actually part of the GraphViz tool-suite, is used for layout — this was seen earlier in Figure 3.

For the main visualization generation in the Execution Interactor, Maple [7] is called from Java in order to generate inequality graphs for sets of constraints. This allowed rapid development of the implementation, although it created a dependency on Maple, as well as restricted the policies to those Maple can support (for example, Maple does not support non-linear inequalities.) However, these limitations are reasonable given the domain of policies. Also, there is a two-level caching mechanism of images to help deal with the speed problem — images are only generated once per policy pair and a given environment (which is encoded into a unique file name), and once an image is loaded in a given application run, it is cached so it does not need to be regenerated or reloaded.

8.1 Handling XACML

The implementation currently deals with real XACML policies by parsing them and translating them into an intermediate format, which is basically a set of constraints that is used as an internal representation of the policy's data. (A note that many of the examples in the images are actually simply handcrafted in the internal representation, which is the reason the combiner states "infeasible" for many of them wrongly.) This conversion is not perfect — the current implementation only supports one rule per policy, since there is no way to currently visualize an OR. However, both

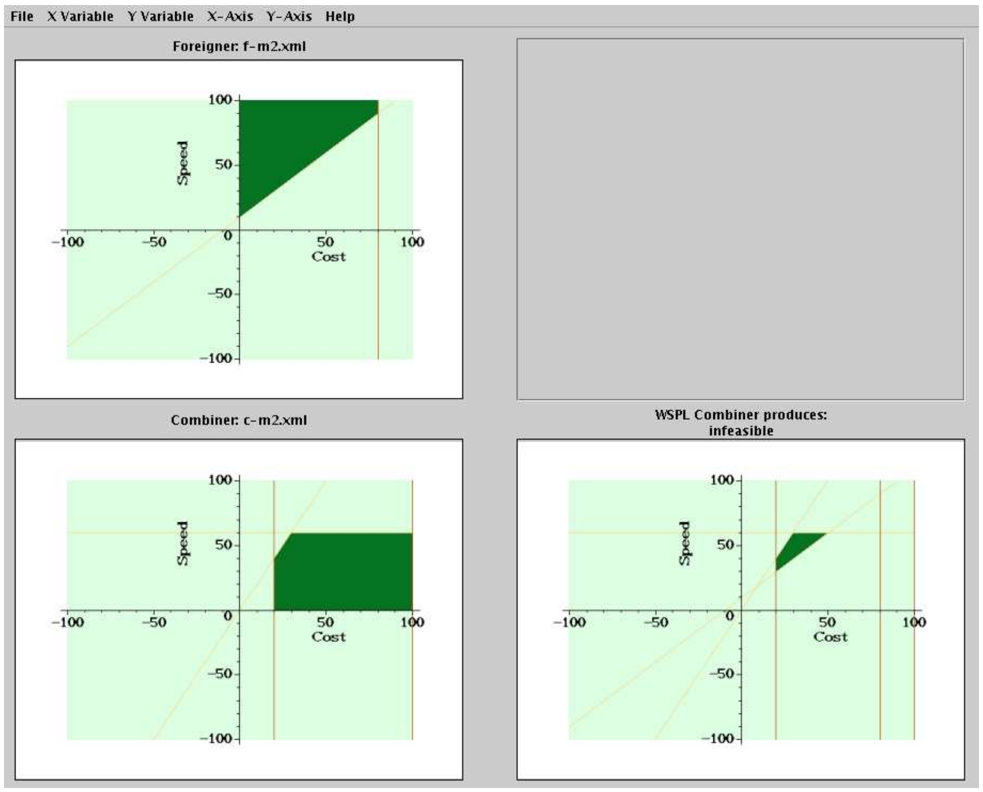

File X Variable Y Variable X-Axis Y-Axis Help
Foreigner: f-m2.xml
Combiner: c-m2.xml
WSPL Combiner produces: infeasible
Speed
Cost
100
50
0
-50
-100

Figure 5: An interesting sample image of the WSPL Execution Interactor than Figure 1. Here, one can see two policies, the end user (foreigner) on the top left and the server (combiner) on the bottom left. The combined policies form a relatively complicated 2D polygon as can be seen on the bottom right. As this shows, even with simple linear policies, complex relationships can arise.

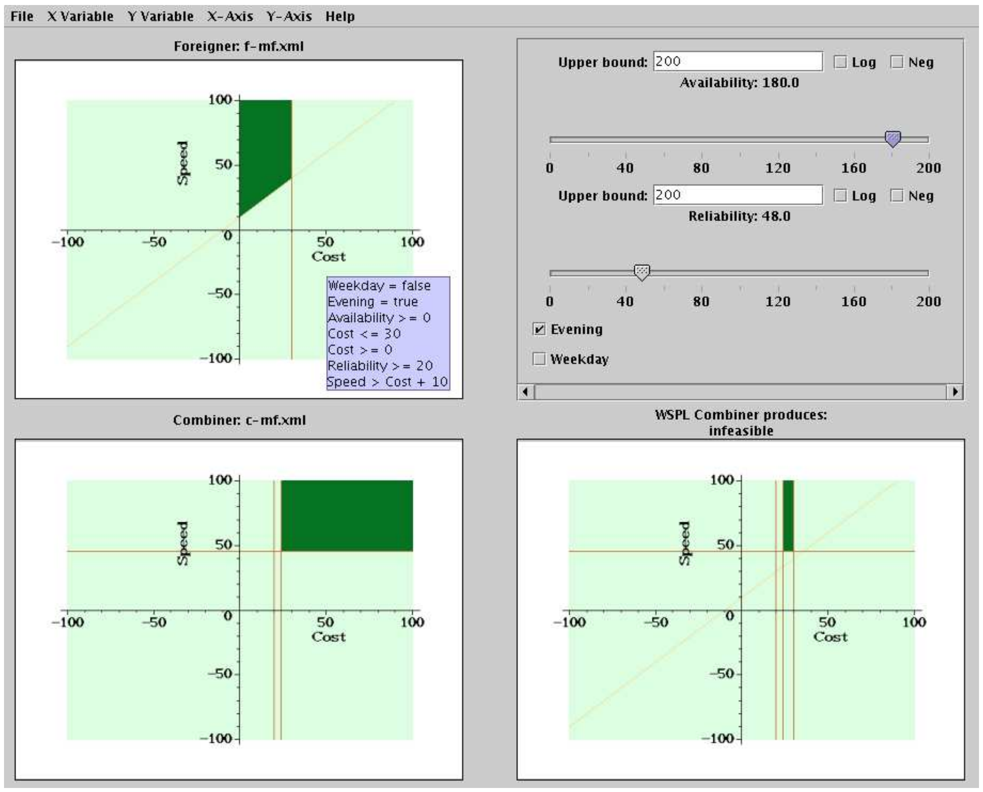
numerical and boolean variables are supported, so with multiple selections of policies with one rule, one could achieve the same affect in a less elegant manner.

8.2 WSPL Interactors

It is worth firstly noting that there is no creation interactor implemented in this project — as Tracy Hadden has already implemented a tool for creating XACML policies, it seemed redundant to reproduce this work.

- Execution Interactor (Figure 5):

  Figure 6 highlights most of the features of the interactor, and is a good place to start discussion, in terms of simple use of the program. After loading the desired policies using the File menu, one must select the desired variables (if the default first two are undesired) to place on the X and Y axes if the policies are defined over more than two attributes. Then, endpoints for the X and Y axes must be selected from the drop-down menus — the current


File X Variable Y Variable X-Axis Y-Axis Help
Foreigner: f-mf.xml
Combiner: c-mf.xml
WSPL Combiner produces: infeasible
Speed
Cost
Log
Neg

Weekday = false
Evening = true
Availability >= 0
Cost <= 30
Cost >= 0
Reliability >= 20
Speed > Cost + 10

Upper bound: 200
Availability: 180.0

Upper bound: 200
Reliability: 48.0

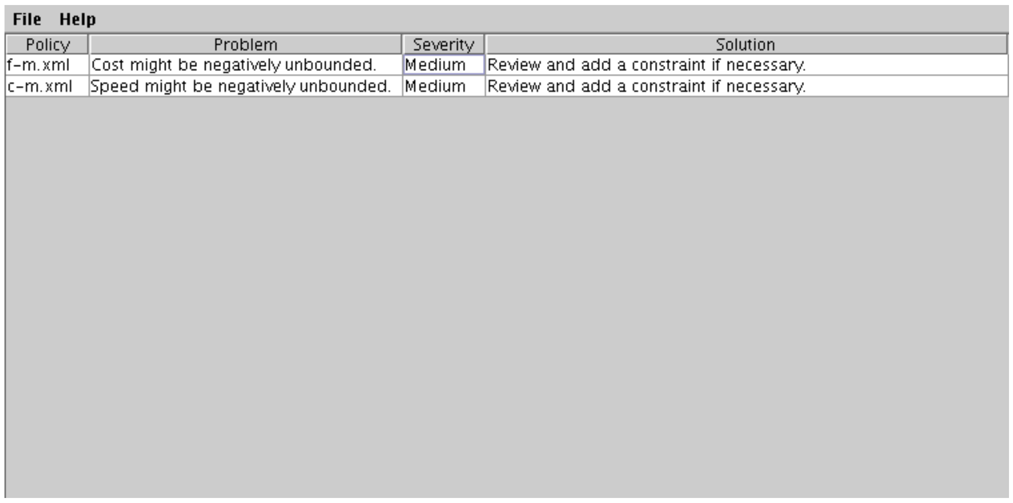
[✓] Evening
[ ] Weekday

Figure 6: A more complex example than Figure 5. Here, one can see the policy display on mouse-over on the top left, as well as the slider widgets for unselected numerical variables that include the slider, checkboxes for selecting log scale and/or negatively valued variables, as well as checkboxes for boolean variables.

options are of the form $[-10^i, 10^i]$ for $0 \leq i \leq 16$, the upper limit being imposed by what Java can represent using a long. As choices for these are made, the graphs will be updated incrementally. The top right contains widgets for manipulating variables. Checkboxes allow for the selection of boolean variables, and a slider "set" allows for the manipulation of numerical variables. One can set the magnitude of the value using the slider, which ranges from 0 to the upper bound selected by the textbox. The negative checkbox allows one to indicate that the value should be interpreted as negative. The log checkbox indicates that the slider value should be interpreted as an exponent in base 10, and will allow values within 5 orders of magnitude under the upper bound currently selected. Lastly, mouse-over of the policy graphs will display the set of constraints that comprise the policy in a text rendering of the internal representation of the constraints. This is useful for both understanding the original XACML policy and to see if anything was lost in the conversion to our internal representation.

- Static Analysis Interactor (Figure 7):

  This relatively simple interactor simply takes a table based format and presents information



Policy	Problem	Severity	Solution
f-m.xml	Cost might be negatively unbounded.	Medium	Review and add a constraint if necessary.
c-m.xml	Speed might be negatively unbounded.	Medium	Review and add a constraint if necessary.

Figure 7: A slightly more complex example of the WSPL Static Analysis Interactor than Figure 2. Here, both policies actually do have variables that are potentially negatively unbounded, and for attributes like cost and speed, this potentially makes for a senseless combined policy.

to the user about potential problems with the policies. The only analysis that the implementation performs for proof-of-concept is to check whether policies are (possibly) negatively unbounded. It doesn't check constraints with more than two variables, so it is possible to falsely trigger this error. However, the implementation is written in such a way that other analyzers which process the policies and return output to display in the table could be easily implemented and composed with the current set of analyzers.

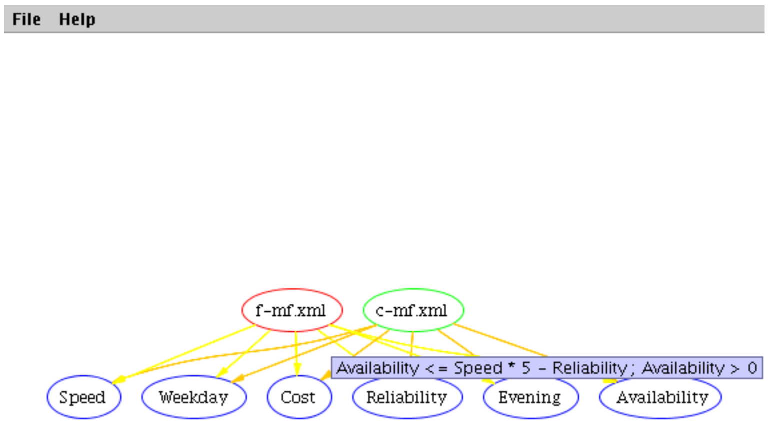
- Item Interactor (Figure 8):

  This interactor is implemented using Grappa for visualization and dot for layout, and displays a bipartite graph of the policies and the attributes on which they are defined. Mousing over the edges, one can see all of the related constraints involved with the (policy, attribute) pair it represents.

8.3 Current Status

The prototype implementation is just under 2000 lines of code, and contains 25 classes and interfaces which comprise the implementation itself and some framework code for potential extensions. It was created using Oracle's JDeveloper IDE.

For more information on the project, including source code, Javadocs, and the latest version of this document, visit `http://www.cs.brown.edu/people/scannell/wsplv.html`.



Availability <= Speed * 5 - Reliability; Availability > 0

Figure 8: A more complex example of the WSPL Item Interactor than Figure 3. One can see that there are many more variables and interactions in these policies, as well as the mouse-over behavior of edges that display all constraints that a given policy has involving a given attribute — in this case, availability.

9 Future Work

There is a lot of room for future development both in this area and on this specific project. First, clearly, a full implementation of the framework and a case study in ATN is an obvious future direction.

Beyond that, fleshing out a few of the issues of the current WSPL visualizer is also an obvious step. Supporting a wider range of policies (explicit OR, multiple rules, etc.) is first on the list, as well as smarter selection of axis ranges are desirable features that were cut due to time constraints. The Item Interactor could benefit from supporting more than two policies being loaded into the program at once (which is a current limitation), as it would allow the user to visualize complicated interactions between many policies and discover dependencies. Writing a custom rendering engine for the constraint graphs to overcome the reliance on Maple, as well as speed up the application, would also be welcome.

The position paper by Papakosta and Burger [9] mentioned earlier outlines one direction of work, which is more focused on classroom presentations and learning environments, and is more advocating the production of good examples for teaching a specific protocol. We suggest that for consumers and administrators of IT, it is equally important, if not more, to focus on building visualizations and interactions for real examples and real exchanges that are occurring on their systems.

10 Conclusion

What has been implemented is a simple yet useful way of determining how WSPL policies will interact, and hint at potential issues in WSPL. Right now, all policy combinations are in the favor of the combiner — the point that is most beneficial to the combiner of the solution space will be chosen. The visualization will hopefully allow an end user to realize this, and modify his policies accordingly.

However, the work described in this paper is only the tip of the iceberg with respect to the number of protocols whose understanding is naturally augmented by flexible and powerful visualizations. WSPL is an extremely simple language that doesn't even have back-and-forth exchanges like ATN, and the visualizer implemented indicates a host of unsolved issues (like supporting OR) that arise with even this simple language. ATN has many other challenges that haven't even been touched — just how to analyze a policy or a set of policies and look for inference holes hasn't even been touched upon here. Distributed credentials call for another layer of complexity to visualize.

Future protocols and algorithms of the more game-theoretic variety (like web agents for online auctions, for example) will add both another layer of complexity and another layer of need — for an end user (or a liable administrator) to feel secure with such entities, it will be necessary to be able to convince her of the safety of its use, and visualizations of interactive protocols will be critical in this endeavor.

11 Acknowledgements

The author thanks Roberto Tamassia at Brown for his support and patience in his role as Masters advisor. The author would also like to thank Danfeng Yao for her WSPL Combiner implementation, Tracy Hadden for her XACML policy creation tool, and Seth Proctor at Sun for his input on WSPL and XACML.

References

[1] Anne Anderson, Seth Proctor, and Simon Godik. XACML profile for Web-services, 2003.

[2] Sean Cannella, Michael Shin, Christian Straub, and Roberto Tamassia. Secure visualization of authentication information: A case study. Unpublished extension to Christian Straub's Honors Thesis, 2003.

[3] Stuart K. Card, Jock D. Mackinlay, and Ben Shneiderman, editors. *Readings in Information Visualization: Using Vision To Think*. Morgan Kaufmann, San Francisco, California, 1999.

[4] Tracy Hadden. XACML and average users: Finding an interface. Brown Independent Study Writeup, 2003.

[5] James Hall, Andrew Moore, Ian Pratt, and Ian Leslie. Multi-protocol visualization: a tool demonstration. In *Proceedings of the ACM SIGCOMM workshop on Models, methods and tools for reproducible network research*, pages 13–22. ACM Press, 2003.

[6] Adam Hess, Jared Jacobson, Hyrum Mills, Ryan Wamsley, Kent E. Seamons, and Bryan Smith. Advanced client/server authentication in TLS.

[7] Maplesoft. Maple application center.

[8] John Mocenigo. Grappa: A Java graph package.

[9] Stella Papakosta and Cora Burger. Generating interactive protocol simulations and visualizations for learning environments, 2001.

[10] Seth Proctor, Steve Hanna, Anne Anderson, and Yassir Elley. XACML.

[11] Michael Shin. ATN visualization demo.

[12] John Stasko, John Domingue, Marc H. Brown, and Blaine A. Price, editors. *Software Visualization: Programming as a Multimedia Experience*. The MIT Press, Cambridge, Massachusetts, 1998.

[13] A. Whitten and J.D. Tygar. Why Johnny can't encrypt: a usability evaluation of PGP 5.0, 1999.

[14] William H. Winsborough and Ninghui Li. Protecting sensitive attributes in automated trust negotiation. In *Proceeding of the ACM workshop on Privacy in the Electronic Society*, pages 41–51. ACM Press, 2002.

[15] William H. Winsborough, Kent E. Seamons, and Vicky E. Jones. Automated trust negotation, January 2000.

[16] Danfeng Yao. WSPL implementation.

[17] Ting Yu, Xiaosong Ma, and Marianne Winslett. PRUNES: an efficient and complete strategy for automated trust negotiation over the internet. In *ACM Conference on Computer and Communications Security*, pages 210–219, 2000.

[18] Chunhua Zhao and Jean Mayo. A TCP/UDP protocol visualization tool: Visual TCP/UDP, 2002.

[19] John A. Zinky and Fredric M. White. Visualizing packet traces. In *Conference proceedings on Communications architectures & protocols*, pages 293–304. ACM Press, 1992.