Pda – The Protocol Derivation Assistant

Matthias Anlauff, Dusko Pavlovic, Kestrel Institute
{ma,dusko}@kestrel.edu

July 17, 2006

Disclaimer

Pda is a graphical editor and support tool for deriving protocols. This document is under construction; some sections are incomplete, others are entirely missing, or in a very early construction phase. Partly this reflects the reality of a rapidly changing interface. Please do not hesitate to contact us to identify problems or inquire about unclear or incomplete documentation.

Contents

1 Introduction

The Protocol derivation assistant (Pda) is a tool to support the incremental derivation of security protocols, together with proofs of their security properties. Just as proof derivations start from definitions and axioms and apply logical transformations, Pda's derivations start from basic protocol components, and then support refinements and transformations to reach the desired protocol. The guiding idea of Pda is to utilize incremental methods of protocol design found in practice, and to support these methods in an open, evolving framework. This idea stands in contrast to attempts to reduce security to a predetermined set of formal rules.

Our goal for the Pda environment is an IDE for production of assured protocols, their derivations, and their security properties, with accompanying proofs. We will over time populate the IDE with a library of protocol components, generic refinements and transformations, and a proof lineage. Of note and importance to some is the fact that the IDE could also be used to identify and prove protocol vulnerabilities and derive witnesses of (attacks on) those vulnerabilities.

2 The Architecture of Pda

Figure **??** sketches the architecture of the Pda tool. The user enters protocol definitions and derivations in the graphical editor, which has a rich set of features to ensure the scalability of the approach. Most prominently, the graphical nodes representing protocols, agents, rules, etc. can be collapsed and expanded as needed, which greatly improves the readability of complex derivation diagrams. While drawing the nodes and edges that make up protocols or derivations, the user gets some live feedback that prevents him/her from adding nodes and edges that are not permitted. For instance, if one side of a send/receive edge has been attached to a state in an agent node, then the user interface makes it impossible to attach the other end of the edge to a state within the same agent. The labels attached to protocols, internal actions, send/receive term, agents and other elements of the derivation are subject to corresponding syntax and semantics rules that are implemented in the parser and static analyzer. If the user makes an error on one of these labels, the graphical editor displays a visual feedback next to the place where the error has been detected. The derivation engine is responsible for performing instantiations and transformation operations, and for providing the result of these operations to the user as new nodes in the graphical editor pane. For example,for an instantiation, the user only enters the definition term for the refined protocol, the process graph of the instance is created automatically by the derivation engine component. All objects involved in the protocol derivations are stored in a database in order to allow for efficient access and update operations. In its current version, the database is built into Pda, but future versions will provide the possibility to use server-based databases.

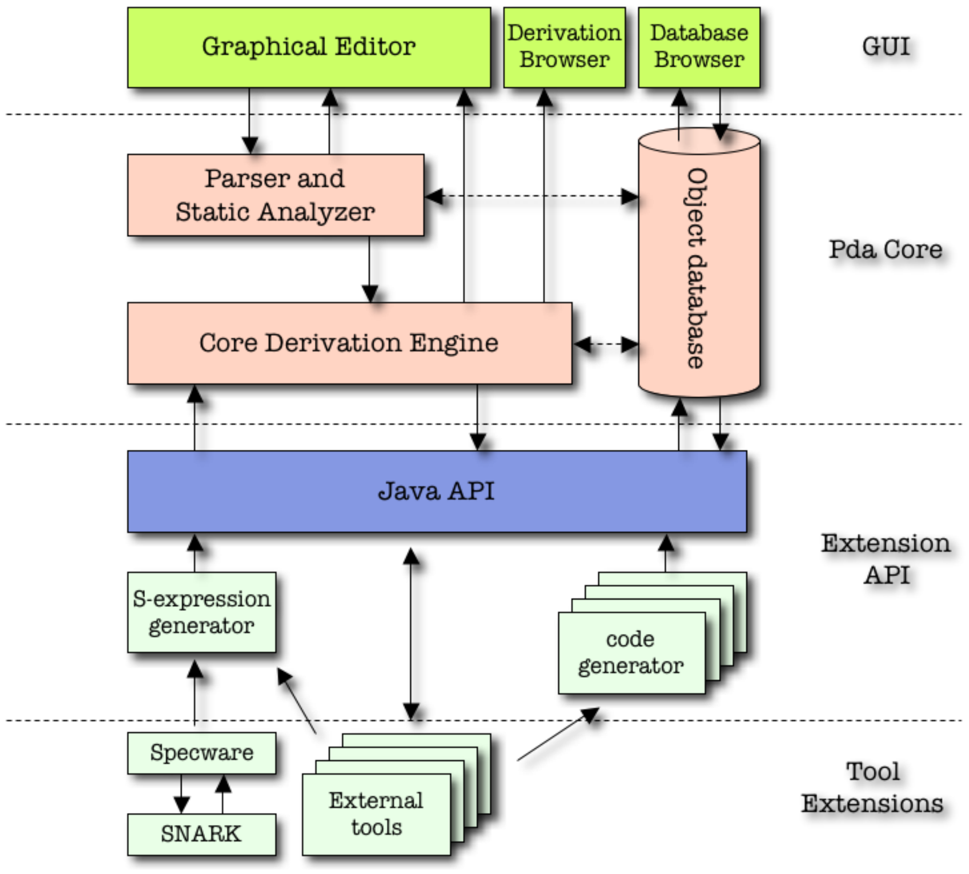


Figure 1: Pda architecture

Pda is also designed to be an integration platform for security-protocol related tools.Pda provides an API that allows Java developers to write plugins for Pda. The API gives access to the internal data structures of the protocols specified by the user and/or loaded into the Pda-database. In order to make Pda also available for extension on a non-Java basis, Pda comes with an S-expression generator that translates the graphics of the protocols and the attached specifications into an S-expression format. The Specware-plugin mentioned earlier makes use of this interface and provides itself a user interface that allows the user to attach model descriptions to protocols. Other code generators can be defined as needed, for instance one for generating executable agent code from the protocol descriptions. Other tools can plug into Pda by either using one of the code generators or by directly using the Java API.

3 User Manual

3.1 Overview

Our objective in the following sections is to introduce the basic elements of the Pda interface as simply as possible. We will use elementary protocols to describe Pda's components and functionality. That should make more readily accessible in the Example section, in which we develop a more complex protocol. Platform dependencies (Windows, Mac, Linux) will be pointed out where necessary, although we handle the main one here, namely *right-click* on Windows and Linux is *ctrl-click* on Mac, but we will just write *right-click*.

Pda essentially works much as a word processor might for constructing (deriving) your next magnum opus. There is a main panel, and collections of buttons, special tool palettes, etc., to support your protocol derivation and analysis. In one way Pda is simpler, since its palettes (on their surface) are not complicated. Pda's interface is possibly also more complicated, since the actions within its toolbars and palettes are accessed through pop-up, context-dependent menus.

One attribute of Pda you will discover through use is its pliability. As you become more expert, you will find that Pda offers you functionality to match your skills. This owes not only to the design of Pda, but also to its nature as an Eclipse plug-in. Tool power goes with tool complexity, but we will try to keep complexities from the Eclipse IDE from distracting your efforts to learn Pda.

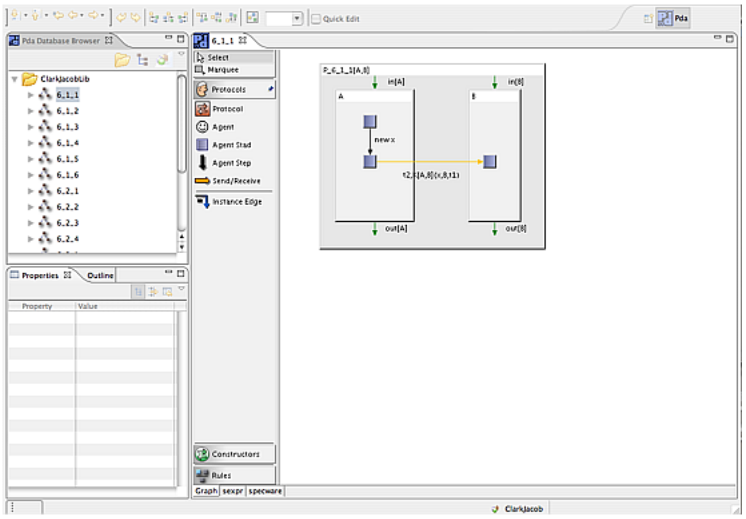
Start your exploration of Pdawith a tour around its main panel in the Main Panel section. Once you have a sense for that, you will be better prepared to understand the menus it contains. Their description can be found in the Menu features section.

3.2 Main panel

Open Pda as you would any application on your machine. During loading, you will be asked if you wish to select the folder in which your workspace is held. Browse your own file system should you need to. You can also set a flag to avoid that dialog each time.

Pda will then ask if you want to start Specware on your machine. The result of your choice (and whether you have Specware installed, should you click *OK*, shows in the colored (red/green) icon at the bottom of the Pda's main panel.

Pda remembers where you were when you last saved and closed the application. The following figure is what you would see if you had the 6_1_1 protocol open for development. (6_1_1 is a name chosen for convenient reference into the ClarkJacob97 listing of protocols.)



The main Pda panel contains four primary components:

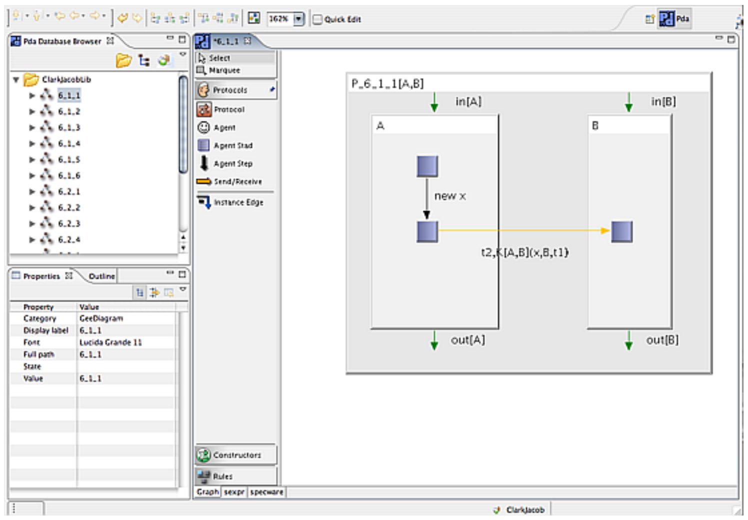
- Pda toolbar
- Database browser pane
- Editing pane
- Properties/Outline pane

Pda toolbar The Pda toolbar is divided into two main groups, each with several components. Where a component has an icon to illustrate the underlying functionality, there is also a cursor-activated descriptor providing a text description. To expose this, simply hold the cursor over the icon (no mouse click).

- *Error-handling controls* The first toolbar group has arrow icons that will eventually be activated to support moving among *error alert* notices, which will be identified by small red, white-crossed circles above protocols showing in the Editing pane. Using these arrows you may navigate among any syntax errors detected by Pda in your protocol specfication. Note: you may also view each error directly by placing the cursor over the alert icons.

- *Layout support tools* The second toolbar group only appears when a protocol is open for editing in the Editing pane. It offers layout support for protocol design actions. Its components are (1) undo/redo, (2) alignment, (3) editor graphics scaling, and (4) a toggle for quick editing actions. In the figure above, the scaling parameter shows as 100% of normal, and Quick Edit as not selected.

The small square colored boxes icon inside in the Application toolbar is your *click-to-zoom* button. Clicking it will zoom the figure in the Editing pane to fill the window to a scaling that keeps the protocol graphic's relative height/width ratio untouched, while filling as much window as possible, as the figure below shows. To return to the unzoomed depiction, you can use the *undo* arrow in the toolbar or find *100%* in the editor graphics scaling measurement in the pull-down menu next to the zoom icon. Alternatively, just click outside the protocol graphic and type the numeral *1*.



Property	Value
Category	GeeDiagram
Display label	6_1_1
Font	Lucida Grande 11
Full path	6_1_1
State	
Value	6_1_1

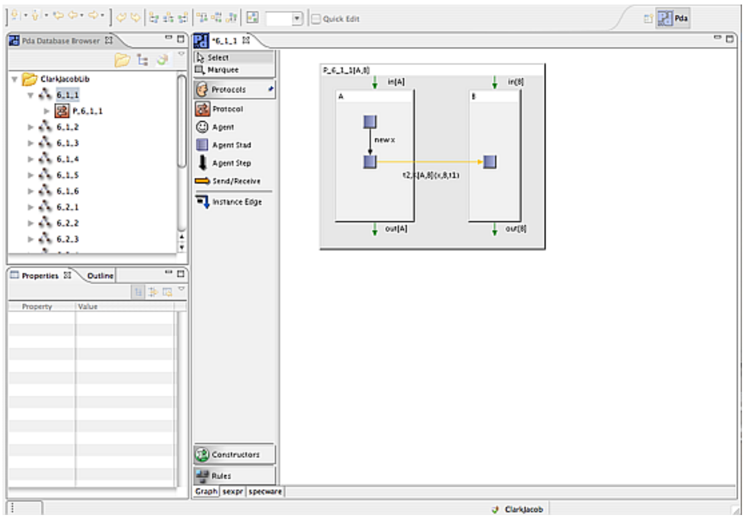
Alignment icons are useful when you want to clean up positions of the internal agent actions in a protocol.

The Quick Edit option is a toggle that changes the default for toolbar selection in the Editing pane. When turned off, after each choice of an element from the Editing pane's toolbar (e.g., Agent Stad), the default selection reverts to the Select arrow. When on, the last toolbar element used is again active for the next mouse click in the Editing pane.

Separated from these two groups, and located at the rightmost end of the Application toolbar, are buttons for choice of perspective. For this overview, we will work only with the default Pda perspective, which is shown selected in the figure (the Pda icon). Additional information on perspectives is available in the *Preparation* Subsection of the *Example* Section.

Notice that the Pda toolbar has a dynamic aspect. When you close all protocols in the Editing pane (see below), the Pda toolbar collapses to just the first two components. In this state, with no protocol open, they are not operative.

Database Browser pane The Database Browser pane is where protocols in your database are organized and displayed. (More precisely, this is where those protocols in your working set within your database are displayed.) The presentation is folder-based, ordered alphabetically, but the actual storage is data-based. In the figure above, you will see displayed the first several protocols in the ClarkJacob library grouping. Each protocol will have derivation constituents. In the case of 6_1_1, the derivation is trivial, i.e., there is just the single derivation component. More complex protocol derivations will have multiple protocol components. These will be listed below their parent in the Browser.



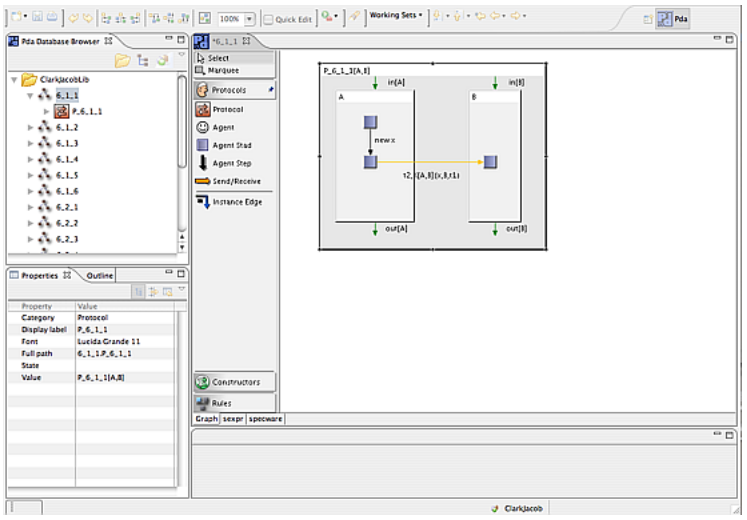
To view a protocol, you right-click the *tree* icon adjacent to the protocol name. Your protocol will open in the protocol graphic editing pane to the right and will also be displayed above that graphic as a tab. This action produced the figure above. Note: double- clicking a protocol component in the Browser will also open the protocol in the Editing pane.

An important part of the Database Browser is the mini-toolbar that appears just below the Pda Database Browser tab. The several icons there are divided into three groups: (1) *folder view*, (2) *derivation view*, and (3) *working set view*. As with the icons in the Application toolbar, there is a cursor-activated descriptor.

The Database Browser display control is particularly important. The first two icons are just a collapse/expand duo. The third one, however, is a pull-down menu that enables control over which folders are displayed in the browser. In particular, notice there the concept of a *working set*, a feature that offers many conveniences. For example, loading protocols into your workspace upon opening Pda can be more efficient if you just load what you need. The same idea applies for exporting a working set, which you might do in order to share your protocol work with a colleague.

Editing pane Perhaps the region where you will place most of your focus is the Editing pane. In the figure above, you see a simple icon that represents a protocol named P_6_1_1. To its left are mini-toolbar buttons that operate within the context of the Protocol editor: Select, Marquee, Protocols (expanded in the figure), Constructors, and Rules. Below the main editor window are two tabs (Graph, sexpr): these control what is displayed in the Editing pane. *Graph* is the default selection; it shows the derivation graph depicted in a agent-arrows format, while *sexpr* shows a lisp-like representation for the flow of actions in the protocol under edit. See the *Tips for Use* Section for more information on *sexpr*.

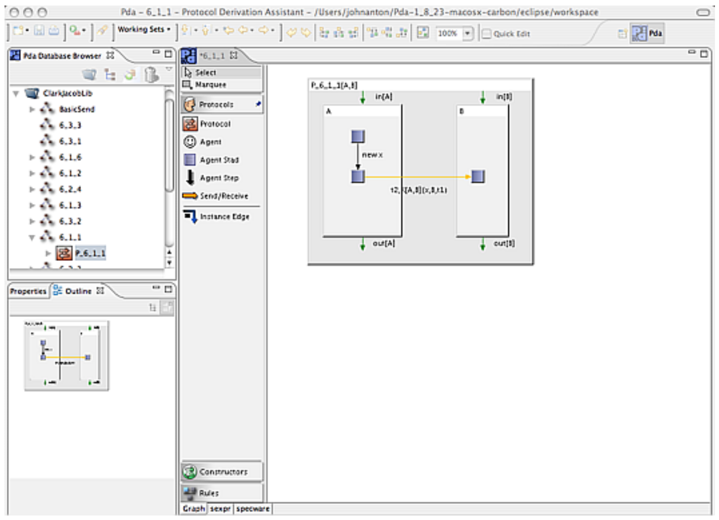
Expanding (double-clicking) P_6_1_1 reveals the picture in the next figure:



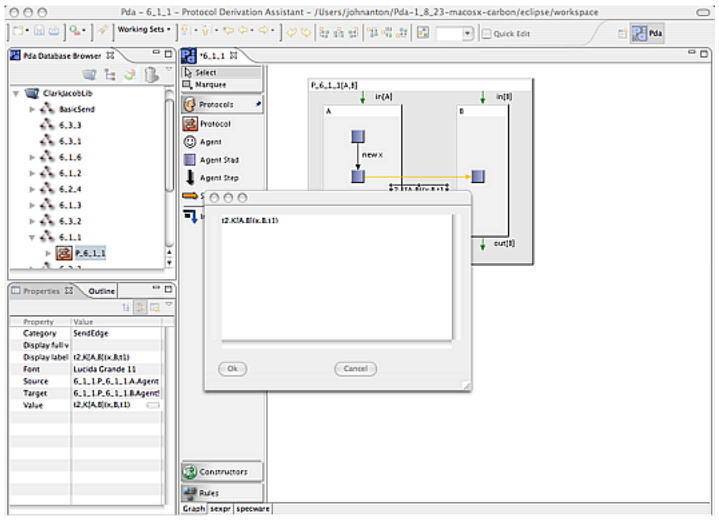
Property	Value
Category	Protocol
Display label	P_6_1_1
Font	Lucida Grande 11
Full path	6_1_1.P_6_1_1
State	
Value	P_6_1_1[A,B]

For convenience the protocol elements in this example are summarized here.(For greater detail, please see the Reference Manual where the Pda protocol derivation language for protocol is presented.) The graph shows two agents, A (Alice) and B (Bob). Alice arrives at the protocol with text elements t1 and t2 already available. She creates a nonce x and sends to Bob a compound message with components t1 and a payload of (x, B, t1) encrypted by a key *K[A,B]* shared between them. The elements in the Protocol Mini Toolbar used to build P_6_1_1 are exposed. More information on their use is located below.

Properties/Outline pane The remaining section of the main panel is tied closely to the protocol editor pane. The next figure shows the Outline tab selected in the Properties/Outline pane. Often a protocol derivation will not conveniently fit in the Editor pane. Move the shaded viewing port in the Outline pane to expose the desired protocol section in the Editor pane.



The Properties pane is useful for seeing a summary of high level properties of each aspect of a protocol, but perhaps even moreso for editing some of them. The next figure shows how to do this. You reach this stage by clicking on the Properties tab, then right-click on a protocol element, select edit from the pop-up contextual window, click on the corresponding value in Properties, and then double-click the small rectangle to its right.


Pda – 6_1_1 – Protocol Derivation Assistant – /Users/johnanton/Pda-1_8_23-macosx-carbon/eclipse/workspace
Pda Database Browser
Properties
Outline
t2,K[A,B](x,B,t1)
Ok
Cancel

Property	Value
Category	SendEdge
Display full v	
Display label	t2,K[A,B](x,B,t1)
Font	Lucida Grande 11
Source	6_1_1.P_6_1_1.A.Agent
Target	6_1_1.P_6_1_1.B.Agent!
Value	t2,K[A,B](x,B,t1)

3.3 Menu features

This section collects explanations of the various menu items throughout Pda. There are two primary types of menus: static and contextual. The former are available by the usual menu pulldown method; the latter by right-clicking on an object in any of the main panel panes described in the *Main panel* section.

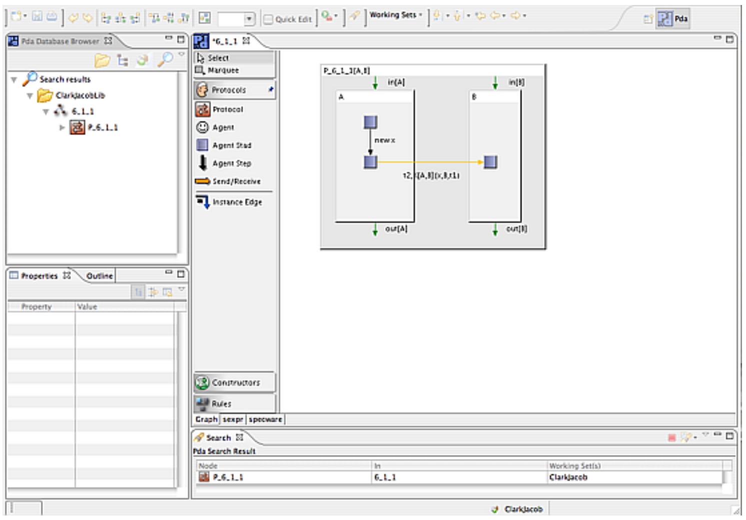
3.3.1 Static menus

The Application menubar contains five menus:Pda, File, Search, Window, Help.

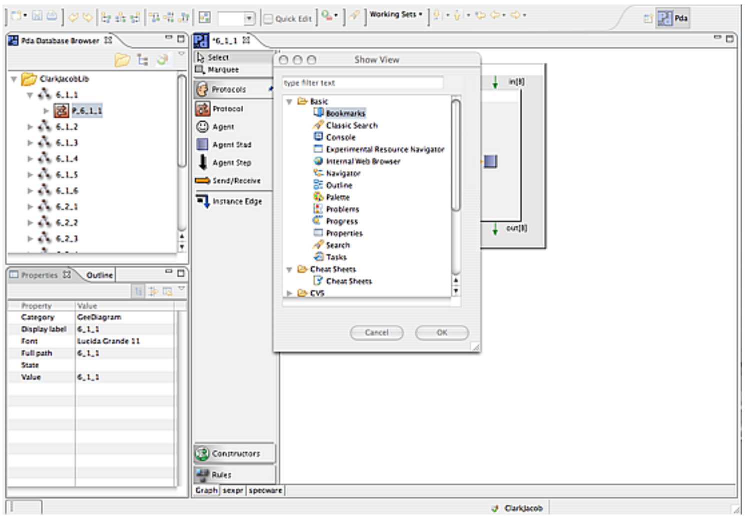
Pda This menu is specific to Mac platforms. The menu items relevant to Pda would be *About Pda* and *Preferences*. At this point, neither item is operational. For Preferences, you go instead to the Window menu, where you will find an item called *Pda Preferences*.

File The primary menu item under File is *Switch Workspace....* This item simply leads to a navigation dialogue in which you pick the folder location of the next workspace you want to use. A workspace is a database containing protocol libraries, in turn possibly partitioned into working sets. Use this menu item to locate the folder in which your desired workspace is contained. You may have different workspaces, but you should keep each in a separate folder.

Search This item has three entries: Search ..., File, and Pda Search, the first of which really contains the other two. Pick Search... and you will see all choices offered to you. The Pda tab is the most interesting, since it enables you to search either your working set or your whole workspace for nodes (protocols, constructors, or rules) or references to nodes. You can see this at work by searching for P_6_1_1. The result is shown in the next figure. Notice that a *Search Glass* icon has been inserted at the third slot in the Browser menubar to indicate that the list of displayed protocols has been cut back to contain only the single one with a reference to the node P_6_1_1. To recover your original workspace display, you can select the *Clear search results*, which is available in the context menu in the search view.

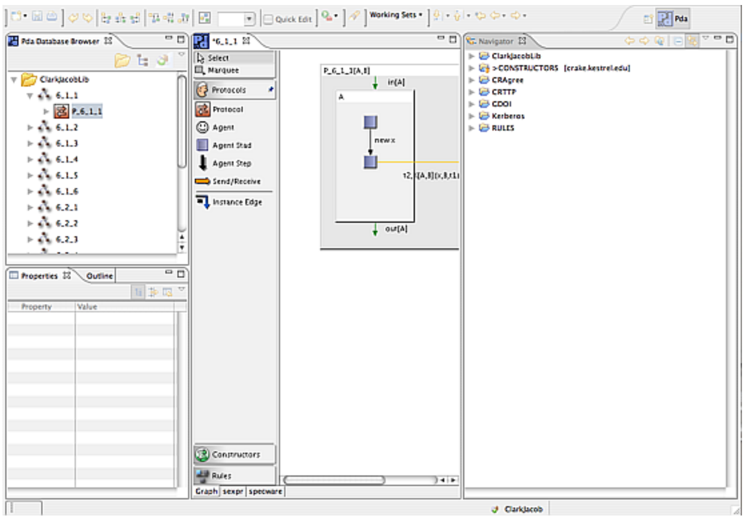


Window This item has four options: Open Perspective, Show View, Reset Perspective, Pda Preferences. Open Perspective has the same functionality as the Other icon at the rightmost end of the Pda toolbar. You will probably just need the Pda option within that dialogue. Show View is more interesting. The next figure displays the various palettes that can be brought up for your support. We will address key palette options in the section on Usage Tips.



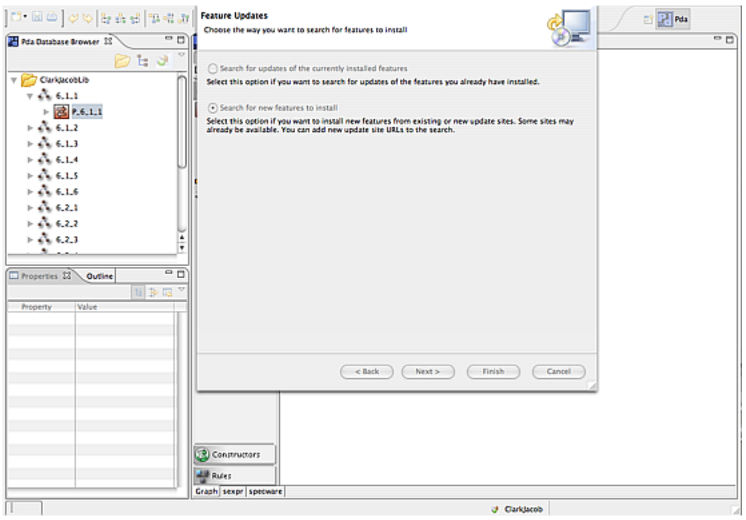
Property	Value
Category	GeeDiagram
Display label	6_1_1
Font	Lucida Grande 11
Full path	6_1_1
State	
Value	6_1_1

For now we just display the result of selecting the Navigation option.



Reset Perspective returns you to the default setting for Pda. Pda Preferences provides a list of options, some self-explanatory. We refer the reader to the Usage Tips section for a discussion of the more advanced options.

Help Search contents is not yet available in Pda. Software updates offers the dialogue shown in the next figure. We advise the reader to pick the second radio button option and then select the Pda filter to check for Pda updates.



Feature Updates

Choose the way you want to search for features to install

( ) Search for updates of the currently installed features

Select this option if you want to search for updates of the features you already have installed.

[✓ Search for new features to install]

Select this option if you want to install new features from existing or new update sites. Some sites may already be available. You can add new update site URLs to the search.

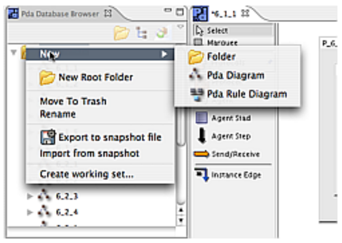
3.3.2 Contextual menus

Contextual menus appear in the Browser pane, the Editing pane, and the Properties pane. To pop-up a contextual menu, right-click in the pane itself or on an object within the pane.

Browser pane menus

Right-clicking in the browser pane produces results that are state dependent. If you have no folder nor protocol selected, your pop-up menu will offer you the option to create a **New Root Folder**. You would accept the option if you wanted to create an additional protocol collection at the top level of your browser hierarchy.

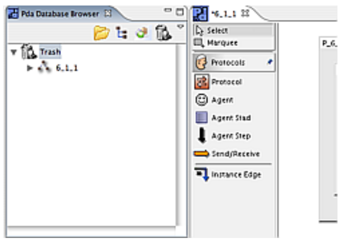
If you right-click with a folder already selected, you see a more elaborate pop-up offering, as shown in the next figure.



New provides a pull-out menu with options for Folder, Pda Diagram, and Pda Rule Diagram. Selecting the first will create a folder within your selected folder, useful for collecting subgroups of protocols. The next two, respectively, create an entry for a new protocol or rule. Pick one and you will be presented with a dialogue to name it.

New Root Folder is the same option offered when no folder is selected.

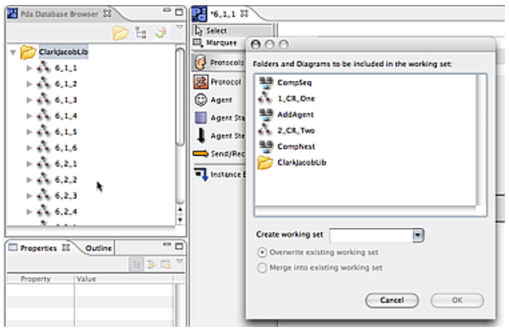
Move to Trash will install a Trash icon in your Browser toolbar and put a selected Folder or protocol or rule into it. If you open the Trash (double-click its icon), select and right-click an item, you are offered the option to restore it. If the Trash is empty, its icon is no longer displayed. If you click on the Trash icon in the Browser toolbar, then right-click on the Trash icon in the Browser, you gain the option of emptying the Trash. As usual, this is not undoable.



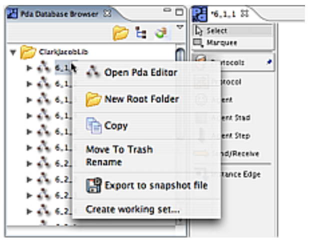
Rename provides a dialogue for changing the name of a folder, protocol, or rule.

Export to snapshot file, resp. **Import from snapshot** are the means by which you export, resp. import an item from your database, for sharing or archiving. Each will enable you to navigate in your platform's file system for the operation.

Create working set allows you to create a filter in your database for convenience in loading and display. The next figure shows the dialogue with options for naming, overwriting, or merging working sets.



If you right-click with a protocol already selected, you see a pop-up menu with only one new option,**Copy**, as shown in the next figure.

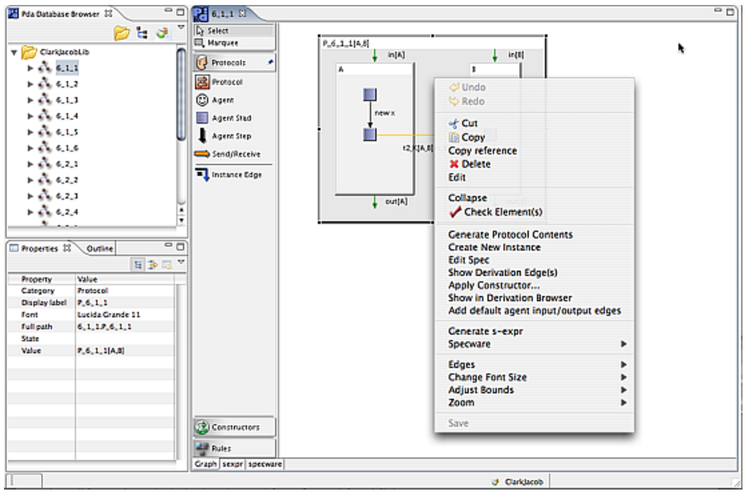


The paired Paste can be executed if you select and right-click a target folder in the Browser.

Editing pane menus

The contextual menus in the Editing pane depend upon whether you wright-click on or in a protocol box or not. If you click outside all protocol boxes, a menu pops up, but only **Zoom** and **Save** result in any action that, in each case,is self-explanatory. The other items will be removed in future versions.

Clicking inside a protocol box, but not on an agent or send/receive arrow, brings up the important menu shown in the next figure.



Property	Value
Category	Protocol
Display label	P_6_1_1
Font	Lucida Grande 11
Full path	6_1_1:P_6_1_1
State	
Value	P_6_1_1[A,B]

Copy reference If you click inside a protocol box, outside of all agents, stads, and arrows, you will see Copy reference in the contextual menu. You can then paste the result into the existing protocol folder or another one that is open. Note, open protocols are shown as tabs above the Editor pane. Copying a reference is analogous to an alias (Mac) or shortcut [pointer] (Win). See the Usage Tips section for more information on this.

Edit Click on an item within a protocol box to open an editing window for that object. The editable items include in and out, stads and arrows. Actually, you edit the contents of the arrow, i.e., its payload description. Editing for in and out and arrows produces the same contextual menu. The entries are self-descriptive. Select Edit to change the value, then click anywhere outside the element to accept the change. Another option after Edit is selected for an object is to click on the object's value in the Properties pane. If a small rectangle appears, clicking that will bring up a larger editing window. See the end of the Main Panel section for an example.

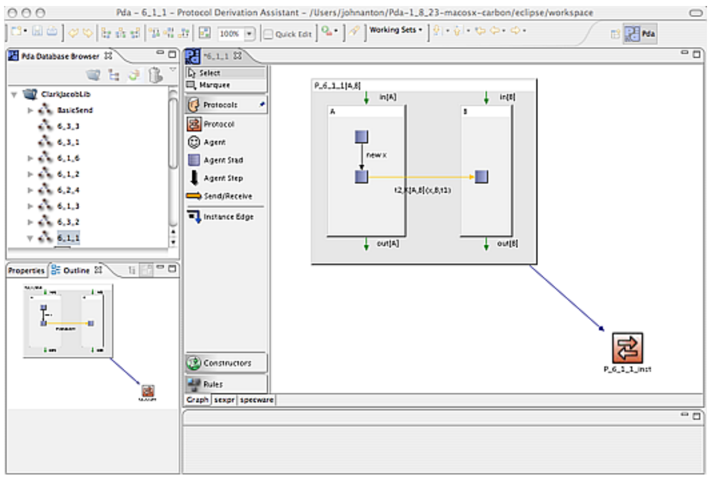
For a stad, the contextual menu contains items that relate to a state in the protocol's evolution (that's what a stad is). Many of the menu items are self-descriptive. The other ones of interest are: *Specware*: this will generate a MetaSlang representation for the stad. *Generate s-expr*: this will generate an s-expression for the stad. *Specware* and *sexpr* data for the collection of stads can be viewed by selecting the specware, resp. sexpr tab below the editing pane.

Collapse This option will collapse the detailed internal view of the protocol to the generic icon form. If your editing pane is full of protocols, you may wish to reduce the display's complexity by collapsing all but the protocol of immediate interest.

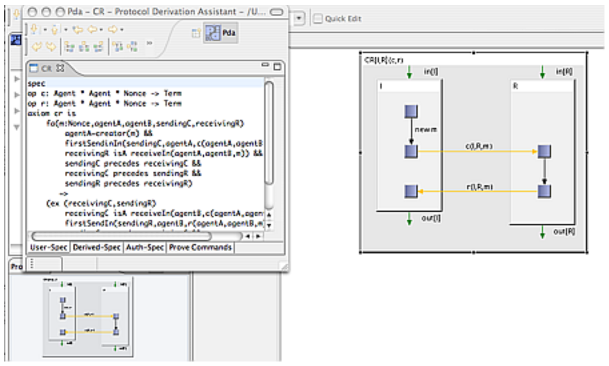
Check Element(s) When you have edited a protocol in the Editing pane, you may wish to check for syntactic correctness without regenerating the whole protocol. Use this option to carry out that function.

Generate Protocol Contents When you edit certain elements in a protocol, the changes may naturally propagate to other items in the current protocol and child nodes in the refinement graph. Although you do not have to regenerate the contents, not doing so affects referential integrity.That is, your protocol will no longer necessarily adhere to structures in its lineage. By the same token, if you want to override a feature in a parent (or higher in the lineage) protocol, do not regenerate the contents you have edited. Opt carefully for override, since this action will immediately update your protocol's definition in your database, with no recourse to *Undo*.

Create New Instance In many protocol derivations you will want a copy of a protocol for describing your refinement . Click on the parent, select Create New Instance, and the result is a child instance as shown in the next figure. Now work your refinement into the new instance. More detail can be found in the Example section.



Edit spec To examine and edit the specs associated with a protocol, use the Edit Spec command. As shown in the next figure this brings up a window with 3 tabs labeled *User-Spec*, *Auth-Spec*, and *Prove Commands*.



```
spec
op c: Agent * Agent * Nonce -> Term
op r: Agent * Agent * Nonce -> Term
axiom cr is
    fa(m:Nonce,agentA,agentB,sendingC,receivingR)
        agentA=creator(m) &&
        firstSendInIn(sendingC,agentA,c(agentA,agentB
        receivingR isA receiveIn(agentA,agentB,m)) &&
        sendingC precedes receivingC &&
        receivingC precedes sendingR &&
        sendingR precedes receivingR)
       =>
    (ex (receivingC,sendingR)
        receivingC isA receiveIn(agentB,c(agentA,agen
        firstSendIn(sendingR,agentB,r(agentA,agentB,m
```

The *User-Spec* is where the user can enter a spec for the functions in the protocol. For example, in the CR protocol (the abstract Challenge-Response protocol) the following spec was entered to characterize the c and r functions.

```
spec
 op c: Agent * Agent * Nonce -> Term
  op r: Agent * Agent * Nonce -> Term
  axiom cr is
  fa(m:Nonce,agentA,agentB,sendingC,receivingR)
  agentA = creator(m) &&
         firstSendIn(sendingC, agentA, c(agentA,agentB,m)) &&
         receivingR isA receiveIn(agentA,r(agentA,agentB,m)) &&
         sendingC precedes receivingR
      =>
  (ex(receivingC,sendingR)
  receivingC isA receiveIn(agentB,c(agentA,agentB,m)) &&
  firstSendIn(sendingR, agentB, r(agentA,agentB,m)) &&
  sendingC precedes receivingC &&
  receivingC precedes sendingR &&
  sendingR precedes receivingR)
endspec
```

The *Auth-Spec* tab contains specs automatically generated by the Pda system. It contains several specs. The first spec is labeled *UserBase* and is a copy of the user spec with an import of the base theory for protocols.

If the protocol is an instance of another protocol, then the *Auth-Spec* contains a spec called *Instance* that contains theorems and conjectures derived from the specs for which this is an instance.

The last spec under the *Auth-Spec* tab is called *Conjectures*. It contains authentication conjectures generated by Pda from the protocol diagram.

Under the *Prove Commands* tab are prove commands of the form *prove conjecture_name* where conjecture_name is a conjecture in the Conjectures spec. The user can ask Pda to try to prove a conjecture by right clicking on a conjecture, moving the mouse down to Specware and then clicking on one of the Prove commands that pops up (these come from the contents of the *Prove Commands* tab). This causes Specware to try to prove the conjecture using Snark.

For more details, please see the Proofs section.

Apply Constructor Constructor design is under review. Design changes are certain, so until then this command should not be used.

Show in Derivation Browser If you right-click on a protocol in the Editing pane, this command will create a derivation browser palette to the right of the pane. All the protocols in your working set will be examined for their dependencies, which the browswer palette then displays. This is the same functionality you will find with the tree icon above the Browser pane.

Add default agent input/output edges This command refers to the input/output annotations at the top/bottom of an agent's slice of the protocol. Select the protocol, choose this menu item and the notations for each Agent Ai will be created: in[Ai]/out[Ai], for each i. If the annotations are already present, even if not the default, no change will be made. One situation where this feature might be convenient is to roll back a user's changes to the default in[]/out[].

Specware This item pulls out to offer the option *Generate spec for protocol.* Selecting this option will have no effect unless a Specware process has been started. (See the Installation Section). If you do have a process running, a formal specification in Specware's language *MetaSlang* will be created.

Generate s-expr Please see the S-Expr Plug-in section for an explanation of this menu item.

Change font size This command can help make a protocol derivation easier to read. Click on any text entry in a protocol to select that component. Then right-click to bring up the contextual menu and select Change font size. You will be provide with a pop-up menu with options to increase or decrease the font size of the selected component.

Adjust bounds ... @@ to be completed

Property pane menus

Fast View, Move, Size, Max/Min, Close

These contextual menu items provide options for altering the location of the panes. You reach the options by clicking in the Properties/Outline pane title area.

@@ to be completed

Copy

If you click in the region within the Properties pane, the item you are pointing to is placed on the Clipboard in the usual manner.

3.4 Example

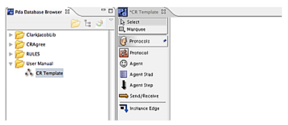
In this section we demonstrate the basic functionality of Pda with a simple derivation. First, however, we set the stage with workspace, perspective, and project steps you need to carry out. Then we will show how to build a two-way authentication protocol out of a one-way authentication protocol using simple derivation steps.

3.4.1 Entering a protocol

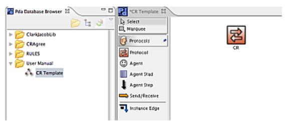
A protocol is a distributed program. To specify a protocol, you must specify a sequence of *actions* to be executed by every participant in the protocol. Actions divide into two groups: *external actions*, i.e., sending and receiving a message and *internal actions*, i.e., generating nonces, keys, hashes, decryption and other local computation. (For agent actions, note that the reserved terms are *if*, *then*, *match* and *new*. For full details on the language to describe actions, functions, and variables, please see the Reference Manual.)

Protocols are entered into Pda with the elements of the Protocol Toolbar. The program of every agent is a linear sequence of *state descriptions* (Stads), with transitions being either *send*, *receive*, or *agent step* (internal compuation).

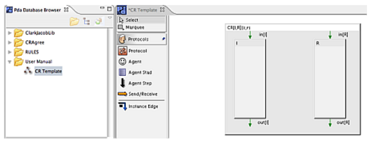
The next figure shows where you are after creating a new protocol diagram called CR Template in your Browser and opening it for edit.



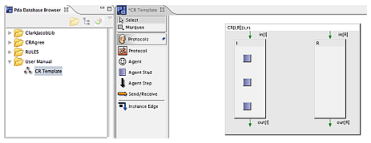
In the Editing pane, select Protocol from the Protocol toolbar, click in the Editing pane, and enter $CR[I, R](c, r)$ to replace the default text (Unnamed_Protocol). Click outside the text area and you will see the next screen.



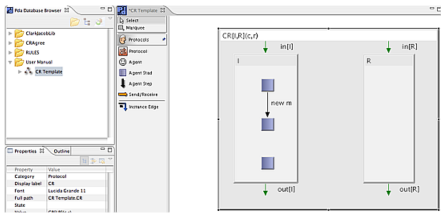
Double-click the collapsed icon to show/edit the protocol, (alternatively, you can right-click to expand it), and you find:



Now click on Agent Stad in the toolbar, then near the top of the Agent I box, then do the same steps again at the middle and then the bottom. You will see the next figure. Notice, after each placement of an Agent Stad, you must reselect the Agent Stad option from the toolbar. That's where Quick Edit can save you time (see the *Main panel* section).

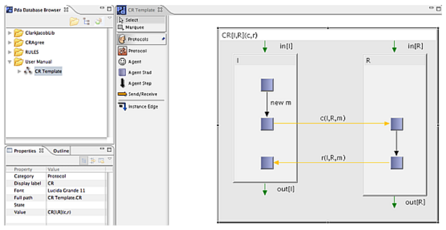


For now, let's use a different aid, namely, *Zoom*. Click inside the protocol, between the two agents, to select it. Then type "z". Return to the toolbar, pick Agent Step. Then click first in the top Agent Stad, release the mouse, position the cursor over the middle Agent Stad, and click/release again. In the window that appears, type "new m", and click outside the agent. You can reposition text by selecting it and using your arrow keys, e.g., to produce the next figure.



Property	Value
Category	Protocol
Display label	CR
Font	Lucida Grande 11
Full path	CR Template.CR
State	

Now use the toolbar items yourself to produce the following two party $CR[I,R](c,r)$ protocol description. Don't forget that you have to reselect toolbar items after each step. For example, when you want to reposition text, you must select Select.



Property	Value
Category	Protocol
Display label	CR
Font	Lucida Grande 11
Full path	CR Template.CR
State	
Value	CR[I,R](c,r)

In this protocol, Agent I (initiator) has the carried out the following sequence of actions: generate a new nonce m, send the message $c(I,R,m)$ to R, receive a message $r(I, R, m)$ from R. Correspondingly, Agent R (responder) receives a message $c(I, R, m)$ from I and sends a message $r(I, R, m)$ to I.

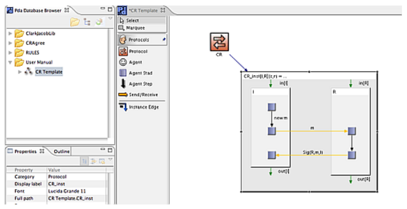
Messages exchanged in a protocol contain concrete cryptographic primitives such as encryption, signature and hash, and function variables such as c and r in this example. The only reserved term for these operations (at this time) is the keyword **new**. Other notations are up to the user. See the section *Conventions* for suggested usage.

When a protocol contains function variables, we say that it is a *protocol template*. In this example, the protocol header $CR[I, R](c, r)$ should be read: CR is a two party protocol (agents are named I and R), using abstract function variables c and r.

3.4.2 Creating an instance

The simplest derivation step is *protocol instantiation*. Using the Create New Instance command (right-click on the CR protocol), some (or all) of the function variables can be refined to concrete primitives or other function variables. For example, we can get a one-way challenge-response protocol using nonces and signature as an instance of $CR[I, R](c, r)$ with the following instantiation: c(x,y,z)=z, r(x,y,z)=Sig(y,z,x), where Sig is a signature function. You enter this instantiation in the editing dialogue that is presented once the instance has been created.

In the resulting protocol $SCR[I, R]$, agent I sends a fresh nonce to R who replies with his signature over the nonce and I's identity.

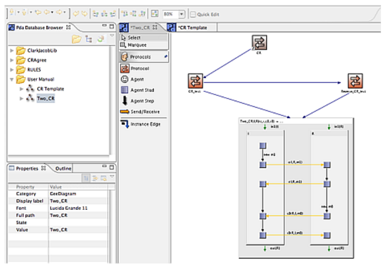


Property	Value
Category	Protocol
Display label	CR_inst
Font	Lucida Grande 11
Full path	CR Template.CR_inst

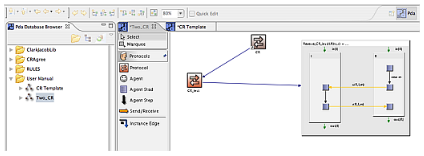
Notice that the figure illustrates having collapsed the Template and expanded the instance.

3.4.3 Sequential composition

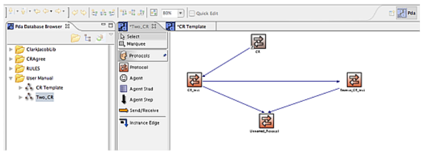
Protocols can be combined using *sequential composition*. In a resulting protocol, the program of each agent is a concatenation of their respective programs from two protocols. For example, protocol $Two\_CR[I, R](c, r, c0, r0)$ is obtained by sequentially composing $CR[I, R](c, r)$ with its reverse copy $Reverse\_CR[I, R](c0, ro)$ (which is in turn obtained by simple instantiation $Reverse\_CR[I, R](c0, ro) = CR[R, I](c0, r0)$). The result is shown in the next figure.



Let's step back to see one way to get to this result. Go to CR Template, open it, select the protocol CR, right-click and pick Copy Reference. Now go to Two_CR and Paste into the Editing pane. Notice that the icon is grayed and has a pointer, indicating that it is a reference instance. Select it, right-click and pick Create New Instance, creating CR_inst. Do the same to CR_inst, but edit its signature to Reverse_CR_inst, with the agent names reversed ([R,I]). At this stage you will see the following figure:



Now use the Protocol toolbar to create another protocol (wait to name it), and the toolbar again to create Instance Edges from the two CR instances above. Now name the composed protocol Two_CR[I,R](c,r,c0,r0)=CR_inst[I,R](c,r);Reverse_CR_inst[I,R](c0,r0). The curly brackets indicate sequential composition of the protocols separated by a semicolon.



Finally, select the composed protocol, right-click and select Generate Protocol Contents.

3.5 Variables

For convenience of use, the diagrammatic protocol representation in Pda is set up to resemble, as much as possible, the informal protocol notation of "arrows-and-messages". On the other hand, one of the main goals of this tool is to support *formal* reasoning about security. To balance between these two goals, for better or for worse, we use a generous set of graphic abbreviations and annotations.

The formal model underlying Pda has evolved through the papers available from www.kestrel.edu/@@. Here we provide a brief summary.

A protocol specification consists of a process, a desired run, and a specification of its security properties. A process is specified as a multiset of agents, which may be the protocol principals, or attackers. Each agent is a program, i.e. a partially ordered multiset of actions. The actions can be internal or external.The internal actions include the usual computational operations, such as assignment and test; and the specific security operations, such as generation of random values.

The external actions can be:

- (send A→B:t), where A and B are constant agent identifiers, and t is a closed term; or
- (recv X→Y:z), where X and Y are agent variables, and z is a term variable

The sender denotes by A and B the purported source and destination of the message, which may or may not be the actual sender and receiver: an attacker can spoof these fields; a router can use them to direct traffic. The receiver denotes by X and Y the claimed source and destination of the received message, and by z its payload.

The desired run of a protocol is an assignment of a unique send action to each receive action. In Pda, this is denoted by drawing arrows between agents. In this way, the process and the desired run are specified together. However, this is a matter of convenience, and it should be clear that processes can have many different runs. In many cases, security means that undesired runs will be detected.

Connecting (send A→B:t) with (recv X→Y:z) in a run has the effect of simultaneously assigning X:=A, Y:=B and z:=t in receiver's environment. In Pda, the source and the destination fields of these two actions are elided from the graphic display whenever they coincide with the actual sender and receiver of the actions linked in the run. Note that this is just a display feature: in the underlying formal model, the source and the destination fields are always present.

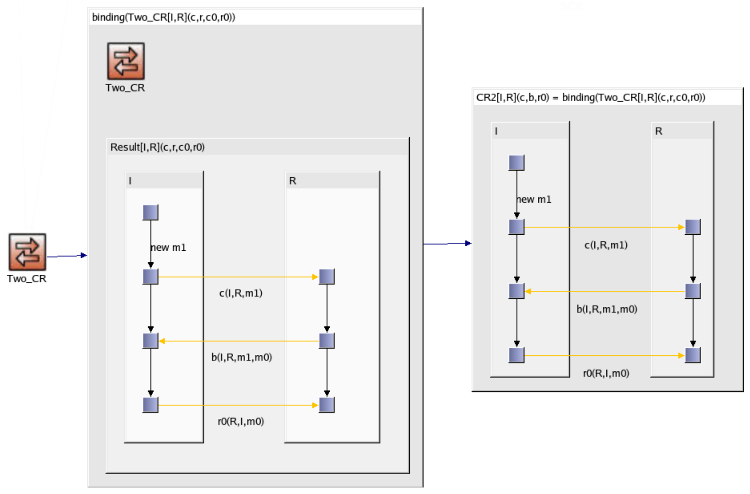
The variables that occur in the receive action are bound to the received values only when the process is run, i.e., dynamically. These variables are necessary to allow specifying statically (at design time), any computations that are to be undertaken at run time. Moreover, while the term structure of the received data (e.g., that it is encrypted by Alice's key) may not be discernible to Bob, he can still manipulate such data, e.g., for passing it forward to a trusted third agent.This is illustrated in the following figure.@@ create figure

@@ need figures and examples

3.6 Constructors

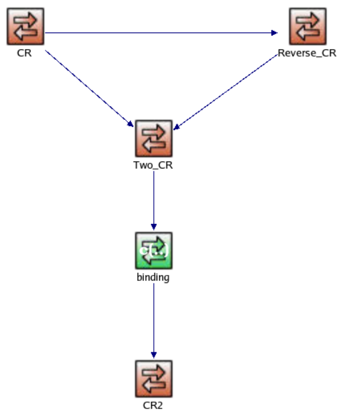
This section describes Constructors through one example. However, at this point, Constructors are being rethought, and subsequent Pda versions will most likely implement them in a different fashion. You are not advised at this point to use them.

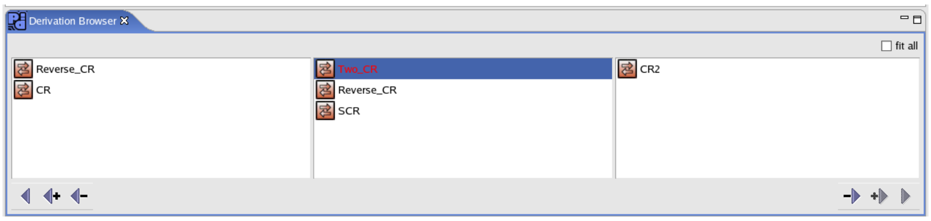
Constructors capture a wide space of protocol transformations. They work like macro recorders. To define a new constructor, you have to specify its parameters by dragging the desired protocols into the constructor window. After that you generate the resulting protocol out of the constructor parameters using the same tools available for protocol construction.



For example, we will define a constructor *binding* that will replace the second and the third message in protocol $Two\_CR[I,R](c,r,c0,r0)$, by $b(I,R,m0,m1)$ obtaining protocol $CR2[I,R](c,b,r0)$.

The structure of the final derivation can be explored in the workspace or with the derivation browser.





Reverse_CR	Two_CR	CR2
CR	Reverse_CR	
	SCR	

3.7 Rules

This description is preliminary. Please expect changes because the design of Rules and Constructors is an active research effort.

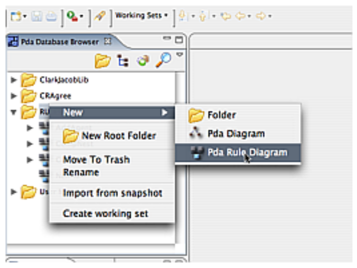
Rules are a more general form of Constructors. The latter take one (or more) protocol(s) as input and provide a form of specialization. (See Section @@ for examples of Constructors.) Rules operate on two or more protocols to combine them in a desired manner. A good model is the case of mutual authentication. Think of such a protocol as either nested or sequential challenge-response (CR) protocols. A rule can be built to compose two instances of a CR protocol into the desired mutual authentication result.

If you open a rule in the Database browser, you will see two additional component entries: *Rule Argument* and *Rule Result*. The idea is that you define a number of rule arguments representing pattern of the protocols you want to apply the rule to, and exactly one rule result, which represents the combination of parts of the rule arguments and possibly new parts.

3.7.1 Applying a rule

3.7.2 Creating a rule

Rules are defined in a special kind of file, a *Pda Rule file*, that can be created as shown in the next figure.



The rule can be applied in a (regular) pda-file by selecting the rule on the palette adjacent to the Editing pane, and placing a reference to it in the diagram. You then draw instance edges from the protocols in the diagram that serve as the rule arguments into the rule symbol.

If all instance edges are attached, double click on the rule symbol and you will get a wizard dialog that allows you to further specify the mappings of the sub-protocols that you have defined in the rule definition with (parts of) the actuals argument protocols. The mapping is done by marking parts of a protocol using a *Sub-Protocol-Marker*, and by connecting each of them with a sub-protocol of the rule argument using a *Sub-Protocol Map Edge*. The labels on the edge can be used to define mappings of terms and agents.

@@ to be completed (need to expand, provide figures, etc.)

3.8 Tips for Use

Here we collect various topics to help you further with Pda.

3.8.1 General tips

Perspective On occasion you may close a pane that you then wish were open. the easiest way to return to the desired view is just to use the PdaPerspective icon in the Pdamenubar.

Arrows When you are placing arrows in a protocol derivation, you may find that Pda's graphics editor are different from others. In Pda, when you have selected either agent step, send/receive, or instance edge, you click/release first in the source box and then in the center of the target box.

Using rules @@ tips for using rules, e.g., make duplicate

Using constructors @@ to be completed

New root folder To create a new root folder, just right-click in the Browser pane to bring up the menu.

Working set @@ to be completed

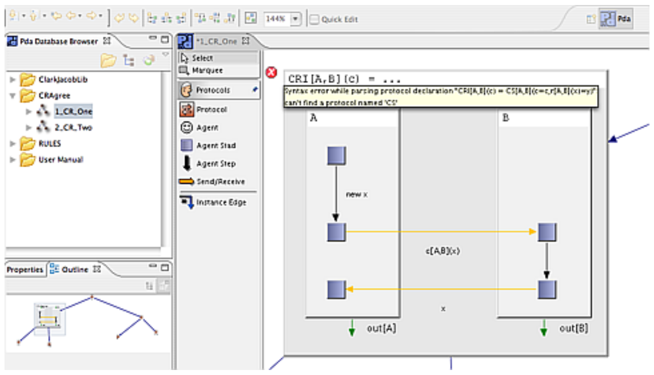
Reference instance To create a reference instance ... @@ to be completed

Undo Pda provides an infinite depth of undo's. Enjoy.

Advanced Pda Preferences @@ to be completed

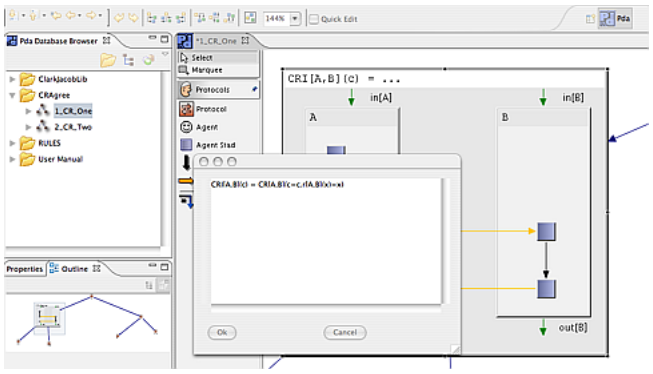
3.8.2 Error alerts

You will make syntactic errors during your editing work in protocol derivation. For example, in the next figure you see that the protocol name CS[A,B] has been mistakenly entered in refining the parent CR[A,B] to create CRI[A,B]. To expose the error report, simply hold the cursor over the red circle (no clicking needed).



```
CRI[A,B](c) = ...
Syntax error while parsing protocol declaration "CRI[A,B](c) = CS[A,B](c=c,r[A,B](x)=y)"
can't find a protocol named 'CS'
```

Notice in the next figure that not only has the protocol name been corrected, but the incorrect response term y has been replaced with the desired term x.



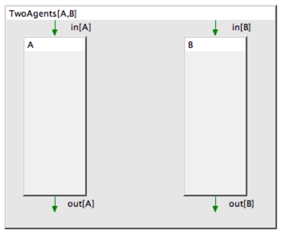
CR[A,B](c) = CR[A,B](c=c,r[A,B](x)=x)

You may navigate among the error alerts by the arrows in the Pda application toolbar. Please see the Main Panel section.

3.8.3 Candidate conventions

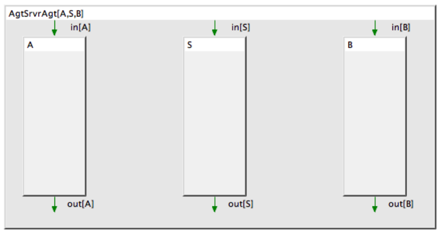
This section provides possible conventions for protocol descriptions in Pda. Users are free, of course, to use their own choices to choose the notation used in a protocol description, as the examples in this manual demonstrate.

- agent names A, B, C, ...



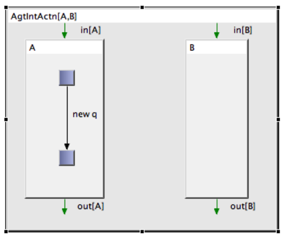
  This image shows the engagement of two agents, A and B. Each agent's internal activities, including accepted input and output produced, will be illustrated in its respective column.

- *server names*: S, S1, S2,...



Protocols can also include a trusted third party, for example a key server for authentication. This image shows the structure to build when a server is part of the protocol. Note that the display of agent activities follows the same order as listed in the argument to the protocol name.

- *agent action names*: *new, if, then, and match* are reserved words in Pda, and are the only action names currently supported.

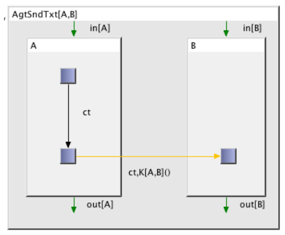


An agent can carry out internal actions, e.g., generation of a nonce, a timestamp, a clear text message, a key, etc. Internal action reserved names include: new, if, then, match. Generic action result names may be labeled q, q1, q2,... The image here shows creation of a new datum labeled q.

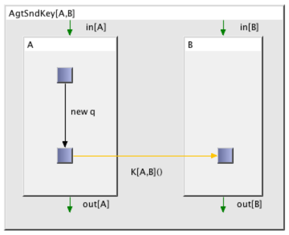
- *nonce names* x, x1, x2, ... Nonces are fresh values used typically in authentication protocols. One form of nonce is a timestamp (see below), but many protocols simply prescribe a value that is randomly generated.

- *timestamp names* ts, ts1, ts2,... Timestamps are an alternative to nonces for authentication. They are used to prevent replay as new of captured messages.

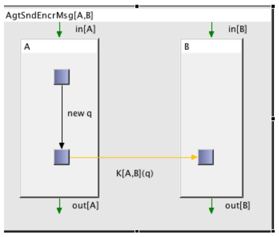
- *clear text names* t, t1, t2,... Clear text can include items such as agent names, public keys, and non-sensitive text messages. The following figure shows Agent A sending clear text along with a shared key to Agent B.



- *key function names* K(), K[A,B](), K[B,S](),... Keys are functions in Pda. The generic key reference is K(), but if the parties sharing the key are named, the convention is to list them as arguments in square brackets, as shown in this figure.



The function notation for arguments is empty when the key is the payload. When a message payload includes encrypted text, it is included in the key function's arugments. The following figure shows this convention.



- *hash function names* H(), H[A,B](), H[B,S](),... Hashes are functions in Pda. The generic hash reference is H().
- *crypto variable names* cv, cv1, cv2,...2
- *names* Protocol names are, of course, up to the designer. For the Clark-Jacob libraries, the numerical naming scheme was chosen for convenient referencing into the paper. Intermediate protocols in a derivation (also, Rule or Constructor) may be used by other derivations, so some attention to their naming may be helpful. Give some thought to picking names, since you will want to recognize reference instances among possibly long lists of protocols in your browser.

3.8.4 Extending Pda

In order to communicate with other tools, Pda uses the Eclipse mechanism of defining *extension points* to provide an interface for tool developers to hook up their tools to Pda. In the current version, only the direction from Pda to an external tool is provided by the interface, but future version will provide for a tighter integration between external tools and Pda.

An example tool plugin for generating s-expressions from protocol descriptions is available at the Pda update site as an additional Eclipse plugin that can be installed on top of Pda. This plugiin also serves for now as the main source of documentation about how to use the interface and about the API for accessing the data provided at the tool plugin interface.

In general, the following things need to be done in order to provide a Pda tool plugin:

1. Create an Eclipse plugin project and extend the edu.kestrel.pda.toolPlugin extension point provided by the edu.kestrel.pda plugin
2. Let the class given in the extension definition extend the class `edu.kestrel.pda.ext.ToolPlugin`.
3. In that class, implement the `processProtocol()` for processing the data structure representing a procotol. The methods `getToolPluginId()`, `getToolPluginName()`, `getProcessProtocolActionLabel()` should also be implemented and just return suitable strings. The latter is used as a label in the context menu of the protocol.
4. If deployed, the `processProtocol()` method will be invoked using the corresponding context menu entry on a protocol in a Pda diagram.

The Java api documentation for the interface classes can be found here.

4 Reference Manual

This section presents a formal description of the language used in Pda for incremental protocol derivation. The language is both graphical and textual; the former addressing nodes & arrows for encapsulating protocol agents and actions; the latter to describe the local agent actions and the payload of send/receive actions.

4.1 The Pda Protocol Derivation Language

In this section, we will define the language of protocol derivations used in the Pda tool. The general idea of the derivational approach is that derivations of protocols go hand in hand with that of proofs of security properties of the protocols. The benefit of this approach is that complex protocols can be derived incrementally from much simpler ones. At the same time the security property proofs are also incrementally developed, so that already proven properties of abstract protocols can be used as theorems in more specific ones. This gives the derivational approach a huge gain compared to other approaches, which try to prove security properties on the final protocols. Because in the latter case the protocol is treated as a monolithic piece of code and/or spec, the complexity of security property proofs can become huge so that it can become infeasible to try proving them with state-of-the-art proving techniques (see [**?**]).

Technically, Pda separates the language of protocol derivations from the language to express security properties and proofs. While the latter one is exchangable in the tool, the language of protocol derivations is pre-defined in order to provide a general framework to define protocols and derivations. In its distributed version, Pda comes with a module for expressing specs and proofs in Specware, Kestrel's state-of-the-art specification language. (Specware itself is not included in the release, but is the framework to communicate to a Specware server process running in the background.) In the following, we will present the Pda protocol derivation language in detail.

4.1.1 Protocol Descriptions

The general idea is that protocols in Pda are described as abstract entities and that certain projection functions exist that further characterize the protocol. For instance, a simple Challenge-Response protocol may be abstractly defined by the term

$$CR[A,B](c,r)$$

which means that the protocol *name* is CR, the *principals* (or *agents*) are A and B, and that the protocol has two function variables c for the challenge and r for the response function. If the developer wishes to define the behavior of this protocol more precisely, he or she can define the actions of the protocol and the partial order between them defining the "desired run" of the protocol. This information is captured by providing a projection function *process* and providing a definition term for the abstractly specified protocol. In Pda, the process description of a protocol is the core projection function for protocols and therefore often referred to as the definition of the protocol.

The Pda protocol language consists of textual as well as graphical elements for describing the entities of the protocol derivations. The graphical elements used are nodes and arrows representing connections between a pair of nodes. Nodes can be nested, i.e., a node can be a child of another node. Graphically this means that the outline of the child node is fully contained within the outline of the parent node. Nodes as well as arrows haves text labels. The following extensions to the EBNF grammar notations are used to represent those relationships:

$\begin{array}{\|c\|}\hline \text{L} \\ \hline \text{N} \\ \hline \end{array}$	represents a graphical node with label L and graphical elements N. N must reduce to a list of nodes and arrows.
$\langle L \mid N_1 \rightarrow N_2 \rangle$	represents a graphical arrow with label L connecting graphical nodes N_1 and N_2.

Using this notation, the description of a protocol is given by the following excerpt of the Pda protocol language:

$$\mathit{ProtocolNode} \quad ::= \quad \begin{array}{|l|}\hline \mathit{ProtDecl} \\ \hline (\mathit{AgentNode} \mid \langle \mathit{SendReceive} \mid \mathit{StateNode} \rightarrow \mathit{StateNode} \rangle)^* \\ \hline \end{array}$$

The label of the protocol node obeys the following syntax rules, where the notation "..." means that the definition of this non-terminal contains more alternatives than shown in this particular rule

ProtDecl	::=	*SimpleProtDecl*
		| ...
SimpleProtDecl	::=	*ProtocolNameIdentifier*
		[*StaticParams*] [*FunctionParams*]
StaticParams	::=	'`[`' *AgentNameIdentifier*
		('`,`' *AgentNameIdentifier*)* '`]`'
FunctionParams	::=	'`(`' *FunctionNameIdentifier*
		('`,`' *FunctionNameIdentifier*)* '`)`'
ProtocolNameIdentifier	::=	⟨ *capitalized identifier* ⟩
AgentNameIdentifier	::=	⟨ *capitalized identifier* ⟩
FunctionNameIdentifier	::=	⟨ *lower case identifier* ⟩

An example of a *SimpleProtDecl* would be the abstract representation of the Challenge-Response protocol as mentioned above: "`CR[A,B](c,r)`".

As specified in the rule for the protocol declaration, the agent nodes are used to specify the principals of the protocol. An agent itself is described by a state machine that specifies the agent's behavior. This state machine is expressed using nodes and arrows between them; the arrows carrying local agent action information.

AgentNode	::=	$\begin{array}{\|l\|}\hline \textit{AgentNameIdentifier} \\ \hline (\textit{StateNode} \mid \\ \langle \textit{AgentAction} \mid \textit{StateNode} \rightarrow \textit{StateNode} \rangle)^* \\ \hline \end{array}$
StateNode	::=	$\begin{array}{\|l\|}\hline \langle \textit{empty} \rangle \\ \hline \langle \textit{empty} \rangle \\ \hline \end{array}$
AgentAction	::=	‘`new`’ *VariableIdentifier*
		| *ActionVariableIdentifier*
		| *VariableIdentifier* ‘`:=`’ *Terms*
		| ‘`if`’ *Term* ‘`then`’ *AgentAction*
VariableIdentifier	::=	⟨ *lower case identifier* ⟩
ActionVariableIdentifier	::=	⟨ *lower case identifier* ⟩

The Pda protocol language defines the following set of terms that can be used in local agent actions as well as the payload of the send/receive messages, the syntax of which is defined further below:

Term	::=	*VariableIdentifier* (’`[`’ *AgentNameIdentifier* ’`]`’)?
		| *FunTerm*
		| ‘`(`’ *Terms* ‘`)`’
		| ⟨ *number* ⟩
		| *Term InfixOp Term*
FunTerm	::=	*FunctionNameIdentifier* [*AgentArgs*] [*Args*]
InfixOp	::=	‘`+`’ | ‘`-`’ | ‘`*`’ | ‘`/`’ | ‘`^`’ | ‘`@`’ | ‘`==`’ | ‘`!=`’ |
AgentArgs	::=	‘`[`’ *AgentNameIdentifier* (‘`,`’ *AgentNameIdentifier*)* ‘`]`’
Args	::=	‘`(`’[*Terms*] ‘`)`’
Terms	::=	*Term* (‘`,`’ *Term*) *

The send/receive arrows represent the message exchange between the agents of the protocol. In their basic format they just contain the payload in form of a term or a list of terms as specified above. In general, the send/receive arrow can also contain data for the sender and/or receiver field of a message; this is used, for instance, to model an intruder who manipulates the original message in order to sham the honest protocol principal. Another use of the explicit sender/receiver field is the “trusted-third-party” protocol, where the agent representing the trusted third party receives and sends the messages with the actual sender/receiver information. The syntax for the send/receive messages is as follows:

SendReceive	::=	[*SenderReceiverAgents* ‘`:`’]
		Terms [‘`|->`’ [*SenderReceiverAgents* ‘`:`’] *Terms*]
SenderReceiverAgents	::=	*AgentNames* ‘`->`’ *AgentNames*
AgentNames	::=	*AgentNameIdentifier* (‘`,`’ *AgentNameIdentifier*)*

4.1.2 Protocol Instantiations

Protocol instantiations can be used to instantiate agent, function, and action parameters of abstract protocols. An example of a protocol instantiation is

```
CRI[A,B](c) = CR[A,B](c=c,r[A,B](x)=x)
```

which means that the protocol `CRI` is defined in terms of `CR` by declaring the `r` parameter to be the identity function. The syntax rules for protocol instantiations are as follows:

ProtDecl	::=	...
		| *ProtInst*
ProtInst	::=	*SimpleProtDecl* ’`=`’ *ProtTerm*
ProtTerm	::=	*ProtTermInst*
		| ...

The protocol term for instantiations is defined as follows:

ProtTermInst	::=	*ProtocolNameIdentifier StaticParams FunctionAndActionInsts*
FunctionAndActionInsts	::=	(’(’ (*FunctionInst* | *ActionInst*) * ’)’)?

A function instantiation provides a definition for a function parameter of the abstract protocol that is being instantiated. There are two variants that can be used to instantiate the functions of an abstract protocol:

- If only function identifiers are used in the instantiation list, then the functions parameters are mapped to the arguments according to the order in the parameter list. For instance, an instantiation

  ```
  CRH[A,B](r1) = CR[A,B](H,r1)
  ```

  would instantiate the `c` parameter of `CR` with `H` and the `r` parameter of `CR` with the local parameter `r1` of `CRH`.

FunctionInst	::=	⟨ *identifier* ⟩
		| ...

- Complete function definitions can be specified in those cases where the simple form can’t be used; each function instantiation resembles a function definition for the parameter on the left-hand side. The function instantiation list must in any case give instantiation for all parameters in the abstract protocol, in either of the formats.

FunctionInst ::= . . .
| *FunTerm* '=' *Term* An example of a function definition of the second form is given in the above instantiation example: `r[A,B](x)=x;`

In either of the forms all function parameters must be mentioned in the argument list of the instantiation.

Action instantiations can be used to replace an action variable attached to a local agent edge in the abstract protocol. The syntax for an action instantiation is

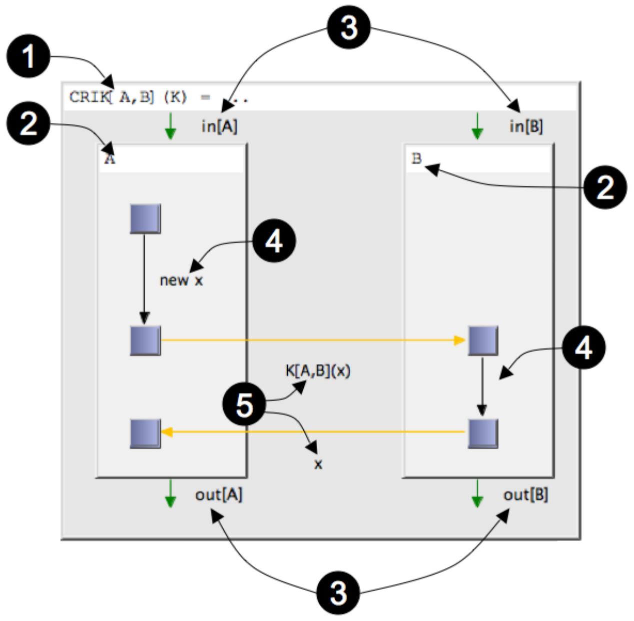
ActionInst ::= ⟨ *identifier* ⟩' =' '{' *AgentAction* '}'

Because, function and action instantiations can occur in any order in the instantion list, the use of an action instantiation implies that use of the second variant for function instantiations, i.e. action instantiations *cannot* be mixed with function instantiations using only identifier and the argument position. An example of an action instantiation would be:

```
a = {new x}
```

4.1.3 Where to use which grammar rules

The following graphic depicts which grammar rules are to be used for which part of a graphical protocol definition in Pda:



No.	Non-terminal symbol	Description
1	*ProtDecl*	Protocol declarations
2	*AgentNameIdentifier*	Agent declarations
3	*Terms*	Input/output terms of agents (optional)
4	*AgentAction*	Local agent actions (might be empty)
5	*SendReceive*	Send/Receive action specs (might be empty)

5 S-Expr Plug-in

This plugin provides a first example of a tool extension of Pda.. The S-Expr Plug-in generates s-expressions from protocols in an Pda diagram and writes them to a file. In the current version, the individual programs of the protocol agents is captured in the generated s-expressions; it is assumed that the connections between the local states of an agents are complete; otherwise the s-expression is not generated correctly.

The S-Expression grammar

The current version of the s-expression grammar that describes the structure of the s-expressions generated by Pda can be found here.

Usage

Specifying the output file: The output file that will be used for writing the s-expression is specified in the Preference page for the plugin. It is accessible from the main menu at *Window* → *Preferences* → *Pda Preferences* → *S-Expr Generation*

Usage: In a Pda diagram, select a protocol node and right-click to get its context menu. The s-expression representing the protocol will be generated into the specified file.

Inspecting the source code

In order to inspect the source code of this Pda tool extension, you have to import the S-Expression plugin as a binary project into your workspace. This allows you to browse the source code in order to get an idea how to write customized Pda tool interfaces.

In more detail, the following steps need to be performed:

1. In the running Pda/Eclipse tool change to the Java perspective and deselect any working sets
2. Run "Import" from the "File" menu
3. Select "Existing Plug-ins and Fragment" and click "Next"
4. Don't change anything on the following screen, click "Next"
5. Select "edu.kestrel.pda.SExprPdaPlugin" on the next screen and click "Add", then "Finish"
6. A project named "edu.kestrel.pda.SExprPdaPlugin" should have been created in your workspace; the source code can be found in the "src" folder of that project.

6 Updates

This link(s) below point to gzip-ed tar-files that can be used to update Pda in case there are difficulties to access the Pda update site from within Eclipse. After downloading the file into a local directory, unpack the file (using the command `tar zxvf file.tgz) in a terminal window` and use the local directory as "New Local Site" in the Eclipse update dialog.

Download updates

Click on the links below to download the update site gzip-ed tar-file:

PdaUpdate 1.8.45

7 Publications

7.1 References

- **A derivational system and compositional logic for security protocols**
  - with A. Datta, A. Derek and J. Mitchell, *J. of Comp. Security 2005, 60 pp.*
- **An encapsulated authentication logic for reasoning about key distribution protocols**
  - with I. Cervesato and C. Meadows, *Proceedings of CSFW 2005 (IEEE), 12 pp.*
- **Deriving, attacking and defending GDOI**
  - with C. Meadows, *Proceedings of ESORICS 2004 (Springer LNCS), 20 pp.*
- **Abstraction and refinement in protocol derivation**
  - with A. Datta, A. Derek and J. Mitchell, *Proceedings of CSFW 2004 (IEEE), 10 pp*
- **Secure protocol composition**
  - with A. Datta and A. Derek and J. Mitchell, *Proceedings of MFPS 2003 (ELNCS); ext. abstract in FMCS 2003 (ACM)*
- **Derivation system for security protocols and its logical formalization**
  - with A. Datta, A. Derek and J. Mitchell, *Proceedings of CSFW 2003 (IEEE)*

- **Compositional logic for protocol correctness**
  - with N. Durgin and J. Mitchell, *J. of Comp. Security 2003; eariler version in CSFW 2001 (IEEE)*
- **Composition and refinement of behavioral specifications**
  - with D. Smith, *ASE 2002 (IEEE)*

see also www.kestrel.edu/home/people/pavlovic

7.2 Related Work (using our framework)

- **A Modular Correctness Proof of TLS and IEEE 802.11i**
  - C. He, M. Sundararajan, A. Datta and A. Derek and J. Mitchell, *to appear in Proceedings of 12th ACM Conference on Computer and Communications Security (ACM 2005)*
- **Compositional Analysis of Contract-Signing Protocols**
  - M. Backes, A. Datta and A. Derek , J. Mitchell and M. Turuani, *Proceedings of 18th IEEE Computer Security Foundations Workshop, pp. 94-110 (IEEE 2005)*
- **Honesty Inferences for Proving Correctness of Security Protocols**
  - K. Hasebe and M. Okada, *Workshop on New Approaches to Software Construction, pp. 45-57 (IEEE 2004)*
- **Non-monotonic Properties for Proving Correctness in a Framework of Compositional Logic**
  - K. Hasebe and M. Okada, *Foundations of Computer Security Workshop, pp. 97-113(IEEE 2004)*
- **Inferences on Honesty in Compositional Logic for Security Analysis**
  - K. Hasebe and M. Okada, *International Symposium on Software Security, Lecture Notes in Computer Science, vol. 3233, pp. 65-86 (Springer 2004)*

A Installation

The section contains a brief description on how to install the Pda software. The contents of the Pda CD can also be accessed online at https://pda.kestrel.edu/pda/CD.

A.1 Installation from the distribution CD

The Pda tool is implemented as a plugin for the public-domain IDE Eclipse. The Pda distribution CD includes a complete installation of Eclipse and the plugins required by the Pda-plugin.

A.1.1 New Installation

If you run Pda on MacOSX, please check whether you have **Java version 1.5.x** installed on your machine. To do that open a terminal and enter the command "java -version". The result must show something like

```
java version "1.5.0_05"
Java(TM) 2 Runtime Environment, Standard Edition (build 1.5.0_05-89)
Java HotSpot(TM) Client VM (build 1.5.0_05-52, mixed mode, sharing)
```

For the windows and linux version, a suitable Java runtime environment is part of the Pda-release.

If you don't have Java installed, you can install it from the CD; the directory "java" contains the suitable installation files for Windows, Linux, and MacOsX architectures.

Installation of Pda on Windows

- Open a file browser and open the "win32" directory on your CD-drive
- Double-click on the zip-file in that directory; this should open the program that is associated with these kind of files (e.g. WinZip).
- Use the zip-tool to extract the contents of the zip-file to some directory, say `C:\Pda`.
- On completion of the extraction the installation is complete. You can exit the zip-tool.
- The executable of the Pda-tool can be found at

  ```
  C:\Pda\Pda
  ```

  (assuming you have extracted the files in `C:\Pda`). You may want to make a desktop shortcut for that file.
- See Section **??** on how to get started with the Pda-tool.

Installation of Pda on Linux and MacOsX

- Open a terminal window.
- Create a new directory where you want the Pda be installed, say `$HOME/pda`.
- cd into the directory and unpack the archive from the cd using the following command, assuming *CD-mount-dir* stands for the directory where your CD drive is mounted (e.g., `/mnt/cdrom` on linux systems):
  - for Linux:

    ```
    tar zxvfp CD-mount-dir/linux-gtk/Pda-x_x_x-linux-gtk.tgz
    ```

  - for MacOsX:

    ```
    tar zxvfp CD-mount-dir/macosx-carbon/Pda-x_x_x-macosx-carbon.tgz
    ```

  (substitute *x_x_x* with the current version number) You should see the files contained in the archive listed on the screen, while they are unpacked into the tool directory.
- After completion of this operation, the installation is complete; the system can by launched by invoking the executable "`$HOME/pda/Pda`" (assuming you have chosen `$HOME/pda` as the tool directory).
- You can call the executable either directly from the command line, make a desktop shortcut, or define an alias in your shell startup file. (e.g., `alias pda=$HOME/pda/Pda` in bournce shell-based environments).
- See Section **??** on how to get started with the Pda-tool.

A.1.2 Updating from earlier Pda-versions

If you have Eclipse 3.x already installed on your machine (which is the case if you have installed an earlier version of Pda before), then you can use the Eclipse-internal update mechanism to install/update Pda. Follow the steps described in Section A.2 except that in step 3 you must select "New Local Site" and navigate to and select the "updates" directory on the CD (e.g., "`D:\updates`" on windows) instead of entering the remote update site.

A.1.3 Installation using existing Eclipse installation

If you have Eclipse version 3.x installed you will need to additionally install the GEF plugin, the graphical editing framework for Eclipse. You can download it from the GEF project site. After that the installation is the same as updating from an earlier Pda-version; follow the instructions above.

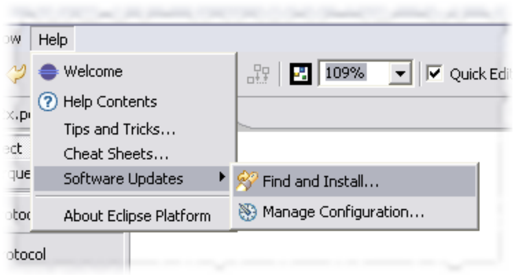
A.2 Download/Installation

The Pda tool is implemented as a plugin for the public-domain IDE Eclipse. If you haven't installed it, you can download and install it from here; Pda requires **Eclipse version 3.x**. You also need to install the **GEF plugin for Eclipse**, which can be downloaded from here.

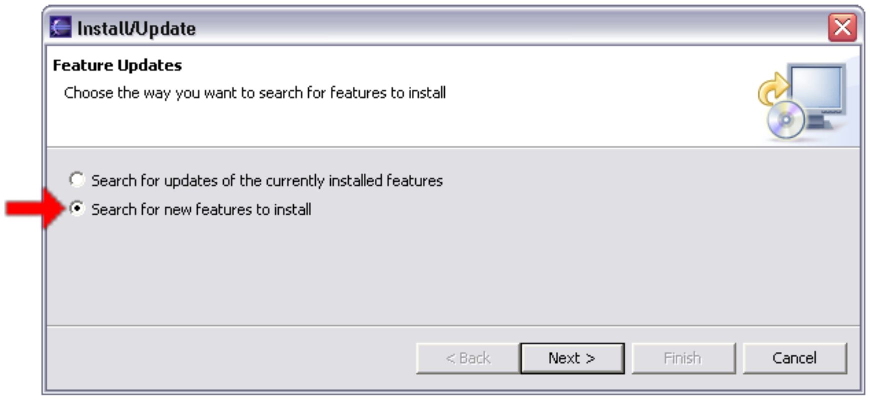
Also, make sure that you have a Java runtime environment/development kit installed; Pda require j2sdk or j2re version 1.4 and above. The Java software can be downloaded, for instance, at here.

The installation of the Pda-plugin can be done from within the running Eclipse-IDE:

1. Start the Eclipse environment and start the update wizard by selecting the menu item "*Help→Software Updates→Find and Install*" as shown here:



2. On the following page, select "Search for new features to install"; select this also, if you are upgrading from an earlier Pda version; click "Next".


Install/Update

Feature Updates

Choose the way you want to search for features to install

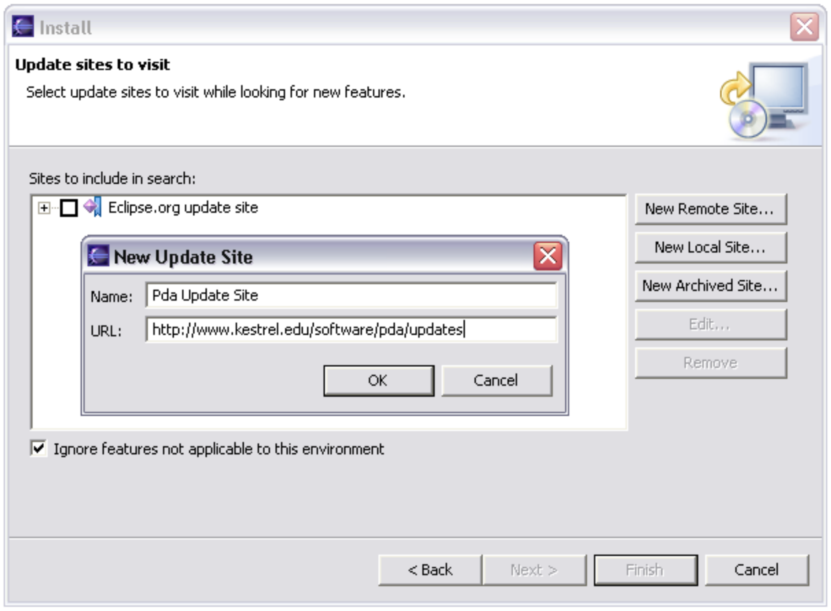
( ) Search for updates of the currently installed features
(•) Search for new features to install

3. On the following page, enter a "New Remote Site" with the following parameters

Name: `Pda Update Site`
URL: `http://www.kestrel.edu/software/pda/updates`

After that an entry for “Pda Update Site” should appear in the list of update sites.

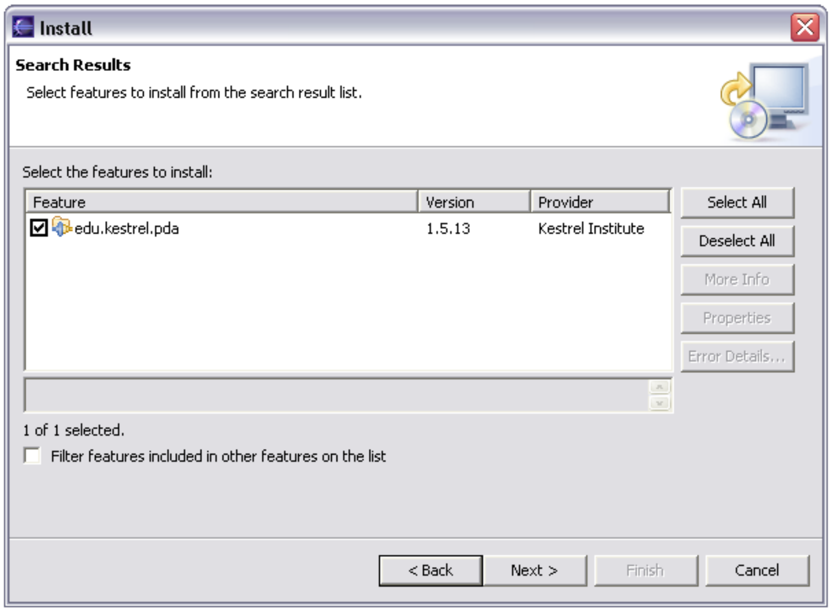
Note: if you are upgrading from an earlier version, you don't need setup the update site, as you already have it.



Name: Pda Update Site
URL: http://www.kestrel.edu/software/pda/updates

☑ Ignore features not applicable to this environment

4. Check the box next to the “Pda Update Site” line and click “Next”
5. On the following page you should see an entry for the latest Pda version. If not it means that you have probably the latest version installed or that your Eclipse configuration is missing a required plugin for Pda.
6. Check the box next to the Pda version you want to install, click “Next”.



Feature	Version	Provider
☑ edu.kestrel.pda	1.5.13	Kestrel Institute

1 of 1 selected.

☐ Filter features included in other features on the list

7. Accept the license agreement and click “Finish” to start the installation.
8. There will be one more screen, where you have to click “Install”.
9. After restarting the Eclipse workbench (done automatically on confirmation) the Pda-Plugin should be ready to use.