Pajé trace file format

B. de Oliveira Stein
Departamento de Eletrônica e Computação
Universidade Federal de Santa Maria - RS, Brazil.
Email: benhur@inf.UFSM.br

J. Chassin de Kergommeaux
Laboratoire Logiciel Systèmes et Réseaux (LSR-IMAG),
681, rue de la Passerelle,
B.P. 72, 38402 St. Martin d’Heres Cedex, France.
Jacques.Chassin-de-Kergommeaux@imag.fr
http://www-lsr.imag.fr/~chassin/

March 22, 2003
Abstract

Pajé is an interactive and scalable trace-based visualization tool which can be used for a large variety of visualizations including performance monitoring of parallel applications, monitoring the execution of processors in a large scale PC cluster or representing the behavior of distributed applications. Users of Pajé can tailor the visualization to their needs, without having to know any insight nor to modify any component of Pajé. This can be done by defining the type hierarchy of objects to be visualized as well as how these objects should be visualized. This feature allows the use of Pajé for a wide variety of visualizations such as the use of resources by applications in a large-size cluster or the behavior of distributed Java applications. This report describes the trace format used by Pajé. Traces include three different kind of informations: definition of the formats of the event, definition of the type hierarchy of the objects to be visualized, definition of the formats of the events of the trace and a set of recorded events, complying with the format definition, to be used to build visualizations according to the type hierarchy.

Keywords: performance debugging, visualization, MPI, pthread, parallel programming, self defined data format.
Contents

1 Introduction 3

2 Extensibility of Pajé 4
   2.1 Introduction ................................................. 4
   2.2 Outline of Pajé .............................................. 5
      2.2.1 ATHAPASCAN: a thread-based parallel programming model ....... 5
      2.2.2 Tracing of parallel programs ................................ 5
      2.2.3 Visualization of threads in Pajé ............................. 5
      2.2.4 Interactivity ............................................. 6
      2.2.5 Scalability: filtering of information and zooming capabilities .... 7
   2.3 Extensibility .................................................. 7
      2.3.1 Modular architecture ....................................... 8
      2.3.2 Flexibility of visualization modules .......................... 8
      2.3.3 Genericity of Pajé ....................................... 9
         2.3.3.1 New data types definition. ............................... 9
         2.3.3.2 Data generation. .................................... 10
   2.4 Conclusion .................................................... 10

3 Definition of type hierarchies and trace event formats 12
   3.1 Introduction ................................................ 12
   3.2 Meta format of Pajé ........................................ 12
   3.3 Events treated by the Pajé simulator .......................... 14
      3.3.1 Definition of types of Containers .......................... 15
      3.3.2 Creation and destruction of containers .................... 15
      3.3.3 Definitions of types of entities ......................... 16
      3.3.4 Creation of visualizable entities ......................... 20
         3.3.4.1 States .................................................. 20
         3.3.4.2 Events ................................................ 20
         3.3.4.3 Variables ............................................. 20
         3.3.4.4 Links .................................................. 20
   3.4 Example ....................................................... 23
   3.5 Visualisation of the activity of the processors of a cluster .......... 24

4 Conclusion ...................................................... 28

Index 31
Chapter 1

Introduction

This report defines the input data format used by the Pajé visualization tool. Pajé is a versatile trace-based visualization tool designed to help performance debugging of large-sized parallel applications. From trace files, recorded during the execution of parallel programs, Pajé builds a graphical representation of the behavior of these programs, to help programmers identify their “performance errors”. The main novelty of Pajé is an original combination of three of the most desirable properties of visualisation tools for parallel programs: extensibility, interactivity and scalability.

Scalability is the ability to represent the execution of parallel programs executing during long periods on large-sized systems; it is provided in Pajé by zooming and filtering functionalities, both in space — ability to synthesize the information originating from several nodes of the system or to zoom in one of these nodes — and in time — possibility to display period of time at various levels of detail. Interactivity is the ability to interrogate visual objects — events, thread states, communications, etc. — to obtain more details or check the source code whose execution produced a given event; it is also the ability to move back and forth in time or to zoom from a synthetic representation to a detailed one or vice versa or to set or reset a filter. Extensibility is the possibility of extend the tool with new functionalities — visual representations, filters, etc. — or to display new programming models. Several characteristics of Pajé contribute to its extensibility: careful modular design, independence of the visualization modules from the programming model.

Key to the ability to build a visual representation of the behavior of parallel programs, developed with various programming models, is the genericity of Pajé: ability to parameterize the tools with a description of what is to be represented and how. This description is provided in the trace file as a hierarchy of the types of objects appearing in the visualization. The format of this description as well as the format of the events of the trace are also described in the trace file. The trace files used as input by Pajé thus contain four categories of data:

1. Description of the format of the generic instructions.
2. Generic instructions, describing the hierarchy of the types of objects appearing in the visualization.
3. Description of the format of the events recorded during the execution of the visualized program.
4. Events recorded during the execution of the program to be visualized.

The aim of this technical report is to describe the Pajé trace data format. The organization of the report is the following. After this introduction, the extensibility and genericity of Pajé are described in detail. The following section defines the Pajé data format and gives an example of use before the conclusion.
Chapter 2

Extensibility of Pajé

2.1 Introduction

The Pajé visualization tool described in this article was initially designed to allow programmers to visualize the executions of parallel programs using a potentially large number of communicating threads (lightweight processes) evolving dynamically. The visualization of the executions is an essential tool to help tuning applications implemented using such a parallel programming model.

Visualizing a large number of threads raises a number of problems such as coping with the lack of space available on the screen to visualize them and understanding such a complex display. The graphical displays of most existing visualization tools for parallel programs [15, 16, 17, 18, 22, 25, 26] show the activity of a fixed number of nodes and inter-nodes communications; it is only possible to represent the activity of a single thread of control on each of the nodes. It is of course conceivable to use these systems to visualize the activity of multi-threaded nodes, representing each thread as a node. In this case, the number of threads should be fairly limited and should not vary during the execution of the program. These visualization tools are therefore not adapted to visualize threads whose number varies continuously and life-time is often short. In addition, these tools do not support the visualization of local thread synchronizations using mutexes or semaphores.

Some tools were designed to display multithreaded programs [14, 27]. However, they support a programming model involving a single level of parallelism within a node, this node being in general a shared-memory multiprocessor. Our programs execute on several nodes: within the same node, threads communicate using synchronization primitives; however, threads executing on different nodes communicate by message passing. Moreover, compared to these systems, Pajé ought to represent a much larger number of objects.

The most innovative feature of Pajé is to combine the characteristics of interactivity and scalability with extensibility. In contrast with passive visualization tools [15, 22] where parallel program entities — communications, changes in processor states, etc. — are displayed as soon as produced and cannot be interrogated, it is possible to inspect all the objects displayed in the current screen and to move back in time, displaying past objects again. Scalability is the ability to cope with a large number of threads. Extensibility is an important characteristic of visualization tools to cope with the evolution of parallel programming interfaces and visualization techniques. Extensibility gives the possibility to extend the environment with new functionalities: processing of new types of traces, adding new graphical displays, visualizing new programming models, etc.

The interactivity and scalability characteristics of Pajé were described in previous articles [5, 6, 8]. This

---

1This chapter is an updated version of the paper published in: *Euro-Par 2000 Parallel Processing. Proc. 6th International Euro-Par Conference* [4]
article focuses on the extensibility characteristics: modular design easing the addition of new modules, semantics independent modules which allow them to be used in a large variety of contexts and especially genericity of the simulator component of Pajé which gives to application programmers the ability to define what they want to visualize and how it must be done.

The organization of this article is the following. The next section summarizes the main functionalities of Pajé. The following section describes the extensibility of Pajé before the conclusion.

2.2 Outline of Pajé

Pajé was initially designed to ease performance debugging of ATHAPASCAN programs by visualizing their executions and because no existing visualization tool could be used to visualize such multi-threaded programs.

2.2.1 ATHAPASCAN: a thread-based parallel programming model

Combining threads and communications is increasingly used to program irregular applications, mask communication or I/O latencies, avoid communication deadlocks, exploit shared-memory parallelism and implement remote memory accesses [9, 10, 13]. The ATHAPASCAN [2] programming model was designed for parallel hardware systems composed of shared-memory multi-processor nodes connected by a communication network. It exploits two levels of parallelism: inter-nodes parallelism and inner parallelism within each of the nodes. The first type of parallelism is exploited by a fixed number of system-level processes while the second type is implemented by a network of communicating threads evolving dynamically. The main functionalities of ATHAPASCAN are dynamic local or remote thread creation and termination, sharing of memory space between the threads of the same node which can synchronize using locks or semaphores, and blocking or non-blocking message-passing communications between non local threads, using ports. Combining the main functionalities of MPI [20] with those of pthread compliant libraries, ATHAPASCAN can be seen as a “thread aware” implementation of MPI.

2.2.2 Tracing of parallel programs

Execution traces are collected during an execution of the observed application, using an instrumented version of the ATHAPASCAN library. A non-intrusive, statistical method is used to estimate a precise global time reference [19]. The events are stored in local event buffers, which are flushed when full to local event files. The collection of events into a single file is only done after the end of the user’s application to avoid interfering with it. Recorded events may contain source code information in order to implement source code click-back — from visualization to source code — and click-forward — from source code to visualization — in Pajé.

2.2.3 Visualization of threads in Pajé

The visualization of the activity of multi-threaded nodes is mainly performed in a diagram combining in a single representation the states and communications of each thread(see figure 2.1).

The horizontal axis represents time while threads are displayed along the vertical axis, grouped by node. The space allocated to each node of the parallel system is dynamically adjusted to the number of visualized threads of this node. Communications are represented by arrows while the states of threads are displayed by rectangles. Colors are used to indicate either the type of a communication, or the activity of a thread. It is not the most compact or scalable representation, but it is very convenient for analyzing detailed
Figure 2.1: Visualization of an ATHAPSCAN program execution

Blocked thread states are represented in clear color; runnable states in a dark color. The smaller window shows the inspection of an event.

threads relationship, load distribution and masking of communication latency. Pajé deals with the scalability problem of this visualization by means of filters, discussed later in section 2.2.5.

The states of semaphores and locks are represented like the states of threads: each possible state is associated with a color, and a rectangle of this color is shown in a position corresponding to the period of time when the semaphore was in this state. Each lock is associated with a color, and a rectangle of this color is drawn close to the thread that holds it (see figure 2.2).

2.2.4 Interactivity

Progresses of the simulation are entirely driven by user-controlled time displacements: at any time during a simulation, it is possible to move forward or backward in time, within the limits of the visualized program execution. In addition, Pajé offers many possible interactions to programmers: displayed objects can be inspected to obtain all the information available for them (see inspection window in figure 2.1), identify related objects or check the corresponding source code. Moving the mouse pointer over the representation of a blocked thread state highlights the corresponding semaphore state, allowing an immediate recognition (see figure 2.2). Similarly, all threads blocked in a semaphore are highlighted when the pointer is moved over the corresponding state of the semaphore. From the visual representation of an event, it is possible to display the corresponding source code line of the parallel application being visualized. Likewise, selecting a line in the source code browser highlights the events that have been generated by this line.
2.2.5 Scalability: filtering of information and zooming capabilities

It is not possible to represent simultaneously all the information that can be deduced from the execution traces. Screen space limitation is not the only reason: part of the information may not be needed all the time or cannot be represented in a graphical way or can have several graphical representations. Pajé offers several filtering and zooming functionalities to help programmers cope with this large amount of information to give users a simplified, abstract view of the data. Accessing more detailed information can amount to exploding a synthetic view into a more detailed view or getting to data that exist but have not been used or are not directly related to the visualization. Figure 2.3 exemplifies one of the filtering facilities provided by Pajé where a single line represents the number of active threads of a node and a pie graph the CPU activity in the time slice selected in the space-time diagram (see [5, 6, 7]) for more details).

2.3 Extensibility

Extensibility is a key property of a visualization tool. The main reason is that a visualization tool being a very complex piece of software costly to implement, its lifetime ought to be as long as possible. This will be possible only if the tool can cope with the evolutions of parallel programming models — since this domain is still evolving rapidly — and of the visualization techniques. Several characteristics of Pajé were designed to provide a high degree of extensibility: modular architecture, flexibility of the visualization modules and genericity of the simulation module.
2.3.1 Modular architecture

To favor extensibility, the architecture of Pajé is a data flow graph of software modules or components (see figure 2.4). It is therefore possible to add a new visualization component or adapt to a change of trace format by changing the trace reader component without changing the remaining of the environment. This architectural choice was inspired by Pablo [22], although the graph of Pajé is not purely data-flow for interactivity reasons: it also includes control-flow information, generated by the visualization modules to process user interactions and triggering the flow of data in the graph (see [5, 6, 7] for more details on the implementation of interactivity in Pajé).

2.3.2 Flexibility of visualization modules

The Pajé visualization components have no dependency whatsoever with any parallel programming model. Prior to any visualization they receive as input the description of the types of the objects to be visualized as well as the relations between these objects and the way these objects ought to be visualized (see figure 2.5). The only constraints are the hierarchical nature of the type relations between the visualized objects and the ability to place each of these objects on the time-scale of the visualization. The hierarchical type description is used by the visualization components to query objects from the preceding components in the graph.

This type description can be changed to adapt to a new programming model (see section 2.3.3) or during a visualization, to change the visual representation of an object upon request from the user. In addition to providing a high versatility for the visualization components, this feature is used by the filtering components. When a filter is dynamically inserted in a data-flow graph — for example between the simulation and visualization components of figure 2.4 to zoom from a detailed visualization to obtain a more global view of the program execution such as figure 2.3 —, it first sends a type description of the hierarchy of objects to be visualized to the following components of the data-flow graph.

The type hierarchies used in Pajé are trees whose leaves are called entities and intermediate nodes containers. Entities are elementary objects that can be displayed such as events, thread states or communications. Containers are higher level objects used to structure the type hierarchy (see figure 2.5). For example:
all events occurring in thread 1 of node 0 belong to the container “thread-1-of-node-0”.

2.3.3 Genericity of Pajé

The modular structure of Pajé as well as the fact that filter and visualization components are independent of any programming model makes it “easy” for tool developers to add a new component or extend an existing one. These characteristics alone would not be sufficient to use Pajé to visualize various programming models if the simulation component were dependent on the programming model: visualizing a new programming model would then require to develop a new simulation component, which is still an important programming effort, reserved for experienced tool developers.

On the contrary, the generic property of Pajé allows application programmers to define what they would like to visualize and how the visualized objects should be represented by Pajé. Instead of being computed by a simulation component, designed for a specific programming model such as ATHAPASCAN, the type hierarchy of the visualized objects (see section 2.3.2) can be defined in a programming model specific file or by inserting several definitions and commands in the trace file (see format in chapter 3). The simulator uses these definitions to build a new data type tree used to relate the objects to be displayed, this tree being passed to the following modules of the data flow graph: filters and visualization components.

2.3.3.1 New data types definition.

In Pajé, it is possible to define new types of containers and new types of data which can be visualized as events, states, links and variables. An “event” is an entity representing an instantaneous action. “States”
of interest are those of containers. A “link” represents some form of connection between a source and a destination container. A “variable” stores the temporal evolution of the successive values of a data associated with a container. Figure 2.6 shows the effect of adding the “threads” container to the type hierarchy of figure 2.5.

2.3.3.2 Data generation.

The trace to be visualized includes the information allowing the creation of containers and entities: events, states (and embedded states using Push and Pop), links — each link being created by one source and one destination call, the coupling between them being performed by the simulator when parameters container, evaluate and key of both source and destination calls match — and change the values of variables.

2.4 Conclusion

Pajé provides solutions to interactively visualize the execution of parallel applications using a varying number of threads communicating by shared memory within each node and by message passing between different nodes. The most original feature of the tool is its unique combination of extensibility, interactivity and scalability properties. Extensibility means that the tool was defined to allow tool developers to add new functionalities or extend existing ones without having to change the rest of the tool. In addition, it is possible to application programmers using the tool to define what they wish to visualize and how this should be represented. To our knowledge such a generic feature was not present in any previous visualization tool for parallel programs executions.

The genericity property of Pajé was used to visualize ATHAPASCAN-I programs executions without having to perform any new development in Pajé. ATHAPASCAN-I is a high level parallel programming model
where parallelism is expressed by asynchronous task creations whose scheduling is performed automatically by the run-time system [3, 11]. The runtime system of ATHAPASCAN-1 is implemented using ATHAPASCAN. By extending the type hierarchy defined for ATHAPASCAN and inserting few instructions to the ATHAPASCAN-1 implementation, it was possible to visualize where the time was spent during ATHAPASCAN-1 computations: computing the user program, managing the task graph or scheduling the user-defined tasks.

Other uses of Pajé include the visualization of distributed applications written in Java [21] or visual monitoring on large sized clusters [12].
Chapter 3

Definition of type hierarchies and trace event formats

3.1 Introduction

A visualization constructed by Pajé is composed of objects organized according to a tree type hierarchy whose nodes are called containers and leaves entities (see §2.3.3). The Pajé data format is self-defined, although it does not comply with the SDDF format used by Pablo [1]. There exists a "meta-format" used to define:

- The format of the instructions defining containers and entities.
- The format of the events recorded during the executions of parallel programs.

These definitions are usually inserted in trace files. They can even be inserted in the observed programs source files, provided that the tracers used to record the events of these programs are able to capture a new definition as a "user-defined" event (see for example figure ??).

Using these definitions, it is possible to define a hierarchy of containers and entities adapted for a given programming model or language. Definitions of type hierarchies as well as instructions and events formats constitute a specialization of the "generic" Pajé visualization tool: it has been used so far to visualize distributed applications written in Java [21] or help to perform system monitoring on large sized clusters [12].

The organization of this chapter is the following. The next section defines the meta format of Pajé used to define the format of type definition instructions and events. The following sections describe how containers and entities are defined and created. The next section is dedicated to the trace events: self definition, recording. The last section of this chapter contains a complete example of use of the Pajé data format.

3.2 Meta format of Pajé

A trace file is composed of events. An event can be seen as a table composed of named fields, as shown in figure 3.1. The first event in the figure can represent the sending of a message containing 320 bytes by process 5 to process 3, containing 320 bytes by process 5 to process 3, containing 320 bytes by process 5 to process 3, containing 320 bytes by process 5 to process 3, 3.233222 seconds after the process started executing. The second event shows that process 5 unblocked at time 5.123002, and that this happened while
executing line 98 of file sync.c. Each event has some fields, each of them composed of a name, a type and a value. Generally, there are lots of similar events in a trace file (lots of “SendMessage” events, all with the same fields); a typical trace file contains thousands of events of tens of different types. Usually, events of the same type have the same fields. It is therefore wise, in order to reduce the trace file size, not to put the information that is common to many events in each of those events. The most common solution is to put only the type of each event and the values of its fields in the trace file. Information on what event types exist and the fields that constitute each of these event types being kept elsewhere. In some trace file formats, this information is hardcoded in the trace generator and trace reader, making the trace structure hard to change in order to incorporate new types of events, new data in existing events or to remove unused or unknown data from those events.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventName</td>
<td>string</td>
<td>SendMessage</td>
</tr>
<tr>
<td>Time</td>
<td>timestamp</td>
<td>3.233222</td>
</tr>
<tr>
<td>ProcessId</td>
<td>integer</td>
<td>5</td>
</tr>
<tr>
<td>Receiver</td>
<td>integer</td>
<td>3</td>
</tr>
<tr>
<td>Size</td>
<td>integer</td>
<td>320</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>EventName</td>
<td>string</td>
<td>UnblockProcess</td>
</tr>
<tr>
<td>Time</td>
<td>timestamp</td>
<td>5.123002</td>
</tr>
<tr>
<td>ProcessId</td>
<td>integer</td>
<td>5</td>
</tr>
<tr>
<td>FileName</td>
<td>string</td>
<td>sync.c</td>
</tr>
<tr>
<td>LineNumber</td>
<td>integer</td>
<td>98</td>
</tr>
</tbody>
</table>

Figure 3.1: Examples of events

A Pajé trace file is self defined, meaning that the event definition information is put inside the trace file itself, much like the SDDF file format used by the Pablo visualization tool [1]. The file is constituted of two parts: the definition of the events at the beginning of the file followed by the events themselves. The definition of events contains the name of each event type and the names and types of each field. The second part of the trace file contains the events, with the values associated to each field, in the same order as in the definition. The correspondence of an event with its definition is made by means of a number, that must be unique for each event description; this number appears in an event definition and at the beginning of each event contained in the trace file.

The event definition part of a Pajé trace file follows the following format:

- all the lines start with a ‘%’ character;
- each event definition starts with a %EventDef line and terminates with a %EndEventDef line;
- the %EventDef line contains the name and the unique number of an event type. The number (an integer) will be used to identify the event in the second part of the trace file. The choice of this number is left to the user. The numbers given in the definitions below (see §?? and §??) are thus arbitrary. The name of the event will be put in a field called “PajeEventName”. There cannot be another field called so. The name is used to identify the type of an event;
- the fields of an event are defined between the %EventDef and the %EndEventDef lines, one field per line, with the name of the field followed by its type (see below).

The structure of the two events of figure 3.1 are shown in figure 3.2.

The type of a field can be one of the following:

date: for fields that represent dates. It’s a double precision floating-point number, usually meaning seconds since program start;

int: for fields containing integer numeric values;
%EventDef SendMessage 21
%  Time     date
%  ProcessId   int
%  Receiver    int
%  Size        int
%EndEventDef

%EventDef UnblockProcess 17
%  Time     date
%  ProcessId   int
%  LineNumber  int
%  FileName   string
%EndEventDef

Figure 3.2: Examples of event definitions
21 3.233222 5 3 320
17 5.123002 5 98 sync.c

Figure 3.3: Examples of events

double: for fields containing floating-point values;

hex: for fields that represent addresses, in hexadecimal;

string: for strings of characters.

named "Time" of type "date", for

The second part of the trace file contains one event per line, whose fields are separated by spaces or tabs, the first field being the number that identifies the event type, followed by the other fields, in the same order that they appear in the definition of the event. Fields of type string must be inside double quotes (") if they contain space or tab characters, or if they are empty.

For example, the two events of figure 3.1 are shown in figure 3.3.

In Pajé, event numbers are used only as a means to find the definition of an event; they are discarded as soon as an event is read. After being read, events are identified by their names. Two different definitions can have the same name (and different numbers), making it possible to have, in the same trace file, two events of the same type containing different fields. We use this feature to optionally include the source file identification in some events. The "UnblockProcess" event in the examples above could also be defined without the fields FileName and LineNumber, for use in places where this information is not known or not necessary.

3.3 Events treated by the Pajé simulator

A Pajé visualization is best described as a typed hierarchy of objects organized as a tree. Elementary objects are the leaves of the tree and called "entities" while intermediate nodes of the tree are named "containers". Entities are the objects that can be visualized in Pajé's space-time diagram, while containers organize the space where those entities are displayed.

Pajé includes a simulator module which builds this hierarchical data structure from the elementary event records of the trace files. Pajé has no predefined containers or entities. Before an entity can be created and visualized, a hierarchy of container and entity types must be defined, and containers must be instantiated.

For example, to visualize the states of threads in a program, one must first define the container types "Program" and "Thread" and the entity type "Thread State". One must also define the possible values that the entities of type "Thread State" can assume (for example, "Executing" and "Blocked"). Then, one must instantiate the program creating a container of type "Program" (called "Thread Testing Program", for example). The threads of the program also have to be instantiated; they are containers of type "Thread".
called for example “Thread 1” and “Thread 2”, and contained in container “Thread Testing Program”. Only then one is able to create visualizable entities of type “Thread State”, by means of events that represent changes in state, contained either in “Thread 1” or “Thread 2”.

The events that the Pajé simulator understands can be divided into four classes:

- events to define types of containers;
- events to define types of entities and possible values that entities can have;
- events to instantiate and destroy containers;
- events to create visualizable entities.

Typically, the events of the first two classes are in the beginning of a trace file, followed by events that instantiate containers, followed by a large number of events creating entities. The simulator does not impose this order, events of these four classes can be mixed in the trace file. The limitation is that an entity or a container cannot be created before its type has been defined and its container created.

The four classes of events are discussed in the following sessions.

### 3.3.1 Definition of types of Containers

Containers types are defined with events named “PajeDefineContainerType”.

**PajeDefineContainerType**

Events of this type (see figure 3.4) must contain the fields “Name” and “ContainerType”. It defines a new container type called “Name”, contained by a previously defined container type “ContainerType” (or the special container type “0” or “/”, if this container type is the top of the container hierarchy). Optionally this event can contain a field “Alias” with an alias name to be used to identify this container. Aliases are usually short strings used when the container name is too big and its use throughout the trace file would increase its size.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string or integer</td>
<td>Name of new container type</td>
</tr>
<tr>
<td>ContainerType</td>
<td>string or integer</td>
<td>Parent container type</td>
</tr>
<tr>
<td>Alias</td>
<td>string or integer</td>
<td>Alternative name of new container type</td>
</tr>
</tbody>
</table>

Figure 3.4: Fields of PajeDefineContainerType event

For the example in section ??, one would need two events (see figure 3.5 to indicate that a “Program” contains “Thread”’s:

### 3.3.2 Creation and destruction of containers

Containers are created using the “PajeCreateContainer” event, and destroyed using the “PajeDestroyContainer” event.
PajeDefineContainerType

This event (see figure 3.6) must have the fields “Time”, “Name”, “Type” and “Container”. Optionally it can have a field named “Alias”. The simulation of this event instantiates, in the simulation time “Time”, a new container named “Name”, of type “Type”, contained in the preexisting container “Container”. The field “Type” must have a value corresponding to the “Name” or “Alias” of a previous PajeDefineContainerType event. The field “Container” must have a value corresponding to the “Name” or “Alias” of a previous PajeCreateContainer event (or “0” or “/”, if on top of the hierarchy). This new container can be referenced in future events by the value of its “Name” or “Alias” (if it has one) field.

PajeCreateContainer

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time of creation of container</td>
</tr>
<tr>
<td>Name</td>
<td>string or integer</td>
<td>Name of new container</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of new container</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Parent of new container</td>
</tr>
<tr>
<td>Alias</td>
<td>string or integer</td>
<td>Alternative name of new container</td>
</tr>
</tbody>
</table>

Figure 3.6: Fields of PajeCreateContainer event

Figure 3.7 shows the events necessary to create the containers “Thread Testing Program” of type “Program” and “Thread 1” and “Thread 2” of type “Thread”, contained by “Thread Testing Program”, from the example in section ??.

PajeDestroyContainer

Containers can be destroyed using the event named “PajeDestroyContainer” with fields “Time”, “Name” and “Type” (see figure 3.8. After simulating this event, the container named (or aliased) “Name” of type “Type” will be marked as being destroyed at time “Time”.

For example, if “Thread 1” finishes execution at time 4.34565, it can be represented by the event in figure 3.9.

3.3.3 Definitions of types of entities

Entities are the leaves of the type hierarchy tree of a Pajé specialization. There exist four types of entities:

- **events**, used to represent an event that happened in a certain point in time, usually displayed as triangles in Pajé’s space-time diagram;
Figure 3.7: Examples of PajeCreateContainer events

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PajeEventName</td>
<td>PajeCreateContainer</td>
</tr>
<tr>
<td>Time</td>
<td>0</td>
</tr>
<tr>
<td>Name</td>
<td>&quot;Thread Testing Program&quot;</td>
</tr>
<tr>
<td>Container</td>
<td>/</td>
</tr>
<tr>
<td>Type</td>
<td>P</td>
</tr>
<tr>
<td>Alias</td>
<td>TTP</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PajeEventName</td>
<td>PajeCreateContainer</td>
</tr>
<tr>
<td>Time</td>
<td>0.986789</td>
</tr>
<tr>
<td>Name</td>
<td>&quot;Thread 1&quot;</td>
</tr>
<tr>
<td>Container</td>
<td>TTP</td>
</tr>
<tr>
<td>Type</td>
<td>T</td>
</tr>
<tr>
<td>Alias</td>
<td>T1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PajeEventName</td>
<td>PajeCreateContainer</td>
</tr>
<tr>
<td>Time</td>
<td>1.012332</td>
</tr>
<tr>
<td>Name</td>
<td>&quot;Thread 2&quot;</td>
</tr>
<tr>
<td>Container</td>
<td>TTP</td>
</tr>
<tr>
<td>Type</td>
<td>T</td>
</tr>
<tr>
<td>Alias</td>
<td>T2</td>
</tr>
</tbody>
</table>

Figure 3.8: Fields of PajeDestroyContainer event

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time of destruction of container</td>
</tr>
<tr>
<td>Name</td>
<td>string or integer</td>
<td>Name of container</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of container</td>
</tr>
</tbody>
</table>

Figure 3.9: Example of PajeDestroyContainer event

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PajeEventName</td>
<td>PajeDestroyContainer</td>
</tr>
<tr>
<td>Time</td>
<td>4.34565</td>
</tr>
<tr>
<td>Name</td>
<td>&quot;Thread 1&quot;</td>
</tr>
<tr>
<td>Type</td>
<td>T</td>
</tr>
</tbody>
</table>
• **states**, used to represent the fact that a certain container was in a determined state during a certain amount of time, usually displayed as rectangles in Pajé’s space-time diagram;

• **links**, used to represent a relation between two containers that started in a certain time and finished in a possibly different time (for example, a communication between two nodes), usually displayed as arrows; and

• **variables**, used to represent the evolution in time of a certain value associated to a container, displayed as graphs in the space-time diagram.

An event of type event, state or link can have a value associated with it, and all possible values must be defined before an event with this value can be created. There are four different events to create an entity type in Pajé, “PajeDefineEventType”, “PajeDefineStateType”, “PajeDefineLinkType” and “PajeDefineVariableType” and one event to define a possible value of an entity, “PajeDefineEntityValue”.

**PajeDefineEventType**

Entities of this new type represent a remarkable type of event recorded during the visualized executions and are displayed as triangles in the space-time diagram. Event types are defined with the “PajeDefineEventType” event. This event (see figure 3.10) contains the fields “Name” and “ContainerType”. It defines a new event type called “Name”, subtype of the previously defined container type “ContainerType”. Optionally it can have a field named “Alias” to have an alternative way to identify this type of entity.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string or integer</td>
<td>Name of new entity type</td>
</tr>
<tr>
<td>ContainerType</td>
<td>string or integer</td>
<td>Type of container of entity</td>
</tr>
<tr>
<td>Alias</td>
<td>string or integer</td>
<td>Alternative name of new entity type</td>
</tr>
</tbody>
</table>

Figure 3.10: Fields of PajeDefineEventType event

**PajeDefineStateType**

Entities of this new type will represent “states”, and are displayed as rectangles in the space-time diagram. The definition contains (see figure 3.11) the fields “Name” and “ContainerType”. Optionally, it can have a field “Alias”.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string or integer</td>
<td>Name of new entity type</td>
</tr>
<tr>
<td>ContainerType</td>
<td>string or integer</td>
<td>Type of container of entity</td>
</tr>
<tr>
<td>Alias</td>
<td>string or integer</td>
<td>Alternative name of new entity type</td>
</tr>
</tbody>
</table>

Figure 3.11: Fields of PajeDefineStateType event

In the example of §3.3.2, the event in figure 3.12 could be used to define the entity type that will represent the states of the threads of the program.
PajeDefineVariableType

Entities of this new type represent variables, whose evolutions are to be visualized as graphs during the execution of parallel programs. Variables are created with the PajeDefineVariableType event (see figure 3.13), containing fields “Name”, “ContainerType” and optionally “Alias”. Their value represent an attribute of a container, whose value (a double) is set by the PajeSetVariable event.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string or integer</td>
<td>Name of new entity type</td>
</tr>
<tr>
<td>ContainerType</td>
<td>string or integer</td>
<td>Type of container of entity</td>
</tr>
<tr>
<td>Alias</td>
<td>string or integer</td>
<td>Alternative name of new entity type</td>
</tr>
</tbody>
</table>

Figure 3.13: Fields of PajeDefineVariableType event

PajeDefineLinkType

Links are used to display a directed link between two containers such as a communication or the identification of a reaction in a container corresponding to an action on another one. The source and destination containers must have a common ancestral in the container hierarchy (identified by “Container” in the events below). Links are usually displayed as arrows.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string or integer</td>
<td>Name of new link type</td>
</tr>
<tr>
<td>ContainerType</td>
<td>string or integer</td>
<td>Type of common ancestral container</td>
</tr>
<tr>
<td>SourceContainerType</td>
<td>string or integer</td>
<td>Type of source container of link</td>
</tr>
<tr>
<td>DestContainerType</td>
<td>string or integer</td>
<td>Type of destination container of link</td>
</tr>
<tr>
<td>Alias</td>
<td>string or integer</td>
<td>Alternative name of new link type</td>
</tr>
</tbody>
</table>

Figure 3.14: Fields of PajeDefineLinkType event

PajeDefineEntityValue

Contains fields “Name”, “EntityName” and optionally “Alias”. Used to give names to the possible values of an entity type. “Alias” will represent the value “Name” that entities of type “EntityName” can have.
### PajeDefineEntityValue

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>string or integer</td>
<td>Value of entity</td>
</tr>
<tr>
<td>EntityType</td>
<td>string or integer</td>
<td>Type of entity that can have this value</td>
</tr>
<tr>
<td>Alias</td>
<td>string or integer</td>
<td>Alternative name of new value</td>
</tr>
</tbody>
</table>

Figure 3.15: Fields of PajeDefineEntityValue event

In the example started in §3.3.2, “Thread State”s can be “Executing” or “Blocked”, as shown in figure 3.16.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PajeEventName</td>
<td>PajeDefineEntityValue</td>
</tr>
<tr>
<td>Name</td>
<td>Executing</td>
</tr>
<tr>
<td>Alias</td>
<td>E</td>
</tr>
<tr>
<td>EntityType</td>
<td>S</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PajeEventName</td>
<td>PajeDefineEntityValue</td>
</tr>
<tr>
<td>Name</td>
<td>Blocked</td>
</tr>
<tr>
<td>Alias</td>
<td>B</td>
</tr>
<tr>
<td>EntityType</td>
<td>S</td>
</tr>
</tbody>
</table>

Figure 3.16: Example of PajeDefineEntityValue event

### 3.3.4 Creation of visualizable entities

There are different events to create entities of each possible type (states, events, variables or links).

#### 3.3.4.1 States

There are events to change a state ("PajeSetState"), to push a state, saving the old state ("PajePushState"), and to pop the previously saved state ("PajePopState").

For example, if "Thread 1" blocks at time 2.34567 and unblocks at time 2.456789, the trace file could contain the events shown in figure 3.18.

#### 3.3.4.2 Events

Events are created with the event named “PajeNewEvent”. Just like states, the values of events must be previously defined by “PajeDefineEntityValue”.

#### 3.3.4.3 Variables

There exist several events to set, add or subtract a value to/from a variable.

#### 3.3.4.4 Links

A link is defined by two events, a “PajeStartLink” and a “PajeEndLink”. These two events are matched and considered to form a link when their respective “Container”, “Value” and “Key” fields are the same.
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the state changed</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of state</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container whose state changed</td>
</tr>
<tr>
<td>Value</td>
<td>string or integer</td>
<td>Value of new state of container</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the state changed</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of state</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container whose state changed</td>
</tr>
<tr>
<td>Value</td>
<td>string or integer</td>
<td>Value of new state of container</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the state changed</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of state</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container whose state changed</td>
</tr>
</tbody>
</table>

Figure 3.17: Fields of state changing events

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PajeEventName</td>
<td>PajeSetState</td>
</tr>
<tr>
<td>Time</td>
<td>2.34567</td>
</tr>
<tr>
<td>Type</td>
<td>&quot;Thread State&quot;</td>
</tr>
<tr>
<td>Container</td>
<td>&quot;Thread 1&quot;</td>
</tr>
<tr>
<td>Value</td>
<td>Blocked</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PajeEventName</td>
<td>PajeSetState</td>
</tr>
<tr>
<td>Time</td>
<td>2.456789</td>
</tr>
<tr>
<td>Type</td>
<td>S</td>
</tr>
<tr>
<td>Container</td>
<td>T1</td>
</tr>
<tr>
<td>Value</td>
<td>E</td>
</tr>
</tbody>
</table>

Figure 3.18: Example of PajeSetState event

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the event happened</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of event</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container that produced event</td>
</tr>
<tr>
<td>Value</td>
<td>string or integer</td>
<td>Value of new event</td>
</tr>
</tbody>
</table>

Figure 3.19: Fields of PajeNewEvent event
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the variable changed value</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of variable</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container whose value changed</td>
</tr>
<tr>
<td>Value</td>
<td>double</td>
<td>New value of variable</td>
</tr>
</tbody>
</table>

**PajeAddVariable**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the variable changed value</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of variable</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container whose value changed</td>
</tr>
<tr>
<td>Value</td>
<td>double</td>
<td>Value to be added to variable</td>
</tr>
</tbody>
</table>

**PajeSubVariable**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the variable changed value</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of variable</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container whose value changed</td>
</tr>
<tr>
<td>Value</td>
<td>double</td>
<td>Value to be subtracted from variable</td>
</tr>
</tbody>
</table>

Figure 3.20: Fields of events that change value of variables

**PajeStartLink**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the link started</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of link</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container that has the link</td>
</tr>
<tr>
<td>SourceContainer</td>
<td>string or integer</td>
<td>Container where link started</td>
</tr>
<tr>
<td>Value</td>
<td>string or integer</td>
<td>Value of link</td>
</tr>
<tr>
<td>Key</td>
<td>string or integer</td>
<td>Used to match to link end</td>
</tr>
</tbody>
</table>

**PajeEndLink**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time</td>
<td>date</td>
<td>Time the link started</td>
</tr>
<tr>
<td>Type</td>
<td>string or integer</td>
<td>Type of link</td>
</tr>
<tr>
<td>Container</td>
<td>string or integer</td>
<td>Container that has the link</td>
</tr>
<tr>
<td>DestContainer</td>
<td>string or integer</td>
<td>Container where link ended</td>
</tr>
<tr>
<td>Value</td>
<td>string or integer</td>
<td>Value of link</td>
</tr>
<tr>
<td>Key</td>
<td>string or integer</td>
<td>Used to match to link start</td>
</tr>
</tbody>
</table>

Figure 3.21: Fields of events that create links
3.4 Example

The whole trace file of the example would be:

```plaintext
%EventDef PajeDefineContainerType 1
%   Alias      string
%   ContainerType string
%   Name       string
%EndEventDef
%EventDef PajeDefineStateType 3
%   Alias      string
%   ContainerType string
%   Name       string
%EndEventDef
%EventDef PajeDefineEntityValue 6
%   Alias      string
%   EntityType string
%   Name       string
%EndEventDef
%EventDef PajeCreateContainer 7
%   Time       date
%   Alias      string
%   Type       string
%   Container  string
%   Name       string
%EndEventDef
%EventDef PajeDestroyContainer 8
%   Time       date
%   Name       string
%   Type       string
%EndEventDef
%EventDef PajeSetState 10
%   Time       date
%   Type       string
%   Container  string
%   Value      string
%EndEventDef
1 P 0 Program
1 T P Thread
3 S T "Thread State"
6 E S Executing
6 B S Blocked
7 0 TTP P 0 "Thread Testing Program"
7 0.986789 T1 T TTP "Thread 1"
10 0.986789 S T1 E
7 1.012332 T2 T TTP "Thread 2"
10 1.012332 S T2 E
```
3.5 Visualisation of the activity of the processors of a cluster

The genericity of Pajé made it possible to visualize the system activity of the processors of a large-sized (200 PEs) cluster of personal computers [12]. Several visualizations were built from the system information available in the /proc pseudo-directory of each PE. It was thus possible to represent the processor and memory load of each PE, the most time consuming process of each PE, etc. Pajé was also used to visualize the reservations of PEs by the users of the cluster, the reservations being done using the PBS system [24] (see figure 3.22). The main bits of the Pajé trace file analyzed to produce this figure are given below.

```plaintext
%EventDef PajeDefineContainerType 1
% Alias string
% ContainerType string
% Name string
%EndEventDef
%EventDef PajeDefineEventType 2
% Alias string
% ContainerType string
% Name string
%EndEventDef
%EventDef PajeDefineStateType 3
% Name string
% ContainerType string
%EndEventDef
%EventDef PajeDefineEntityValue 6
% Name string
% EntityType string
%EndEventDef
%EventDef PajeCreateContainer 7
% Time date
% Alias string
% Type string
% Container string
% Name string
%EndEventDef
%EventDef PajeDestroyContainer 8
% Time date
% Name string
% Type string
```
Figure 3.22: Scheduling of jobs on a large-sized cluster

%EndEventDef
%EventDef PajeSetState 10
% Time date
% Type string
% Container string
% Value string
%EndEventDef
1 MG 0 M-Grappe
1 G MG Grappe
1 M G Machine
1 CPU M Processeur
3 pbs-task CPU
7 7 MG1 MG 0 M-grappe_1
7 7 G1 G MG1 Grappe_1
6 nobody pbs-task
6 chapron pbs-task
6 charao pbs-task
6 fchaussum pbs-task
6 feliot pbs-task
6 guilloud pbs-task
6 gustavo pbs-task
6 leblanc pbs-task
6 maillard pbs-task
6 mpillon pbs-task
6 paugerat pbs-task
6 plumejea pbs-task
6 romagnol pbs-task
6 sder pbs-task
7 8 M_icluster11 M G1 M_icluster11
7 8 P_icluster11 CPU M_icluster11 P_icluster11
7 8 M_icluster21 M G1 M_icluster21
7 8 P_icluster21 CPU M_icluster21 P_icluster21
7 8 M_icluster31 M G1 M_icluster31
7 8 P_icluster31 CPU M_icluster31 P_icluster31
7 8 M_icluster41 M G1 M_icluster41

[...]
| 8 1465976 | G1 G |
| 8 1465976 | MG1 MG |
Chapter 4

Conclusion

Pajé is a versatile visualization tool which can be used in a large variety of contexts. This report describes the data format used by Pajé. Pajé being trace-based, the data actually used for the visualisation is to be presented as a set of execution events. In addition, a description of the type hierarchy of the visual objects needs to be included in the data (trace) file. Both the formats of the type hierarchy description and of the events being self defined, there also need to be a definition of these formats in the input data (trace) file used by Pajé.

The versatility property has been used so far to visualize a distributed Java application and the activity of the nodes of a large-sized cluster of PCs.

To enlarge the applicability of Pajé, a translator from traces produced by Tau [23] into the Pajé format is currently being implemented.
Bibliography


Index

container, 8, 10, 12

date, 13

EndEventDef, 13
entity, 8, 10, 12
event, 9, 16
EventDef, 13

link, 10, 18

PajeAddVariable, 20
PajeCreateContainer, 16
PajeDefineContainerType, 15
PajeDefineEntityValue, 19
PajeDefineEventType, 18
PajeDefineLinkType, 19
PajeDefineStateType, 18
PajeDefineVariableType, 19
PajeDestroyContainer, 16
PajeEndLink, 20
PajeNewEvent, 20
PajePopState, 20
PajePushState, 20
PajeSetState, 20
PajeSetVariable, 20
PajeStartLink, 20
PajeSubVariable, 20

SDDF, 12
state, 9, 18

trace file, 3
type hierarchy, 8, 10

variable, 10, 18