si: a pointer to a SSAPindication structure, which is updated only if the call fails.

If the call to SMinSyncResponse is successful, then the minor synchronization has been completed. Otherwise the SSAPabort structure contained in the SSAPindication parameter si contains the reason for failure.

### ReSynchronization

To resynchronize the connection to a known state, the SReSyncRequest is used (which corresponds to the S-RESYNCHRONIZE.REQUEST action).

```
int     SReSyncRequest (sd, type, ssn, settings, data, cc, si)
int     sd;
int     type,
        settings;
long    ssn;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

sd: the session-descriptor;

type: the type of resynchronization requested (either SYNC_RESTART, SYNC_ABANDON, or SYNC_SET);

ssn: the serial-number to resynchronize to;

settings: the new token settings;

data/cc: any additional data (and the length of that data, which may not exceed SN_SIZE octets); and,

si: a pointer to a SSAPindication structure, which is updated only if the call fails.

If the call to SReSyncRequest is successful, then the resynchronization has been queued for the remote user. When the S-RESYNCHRONIZE.CONFIRMATION event is received, the resynchronization is complete. Otherwise

the SSAPabort structure contained in the SSAPindication parameter si contains the reason for failure.

Upon receiving a S-RESYNCHRONIZE.INDICATION event, the user is required to generate a S-RESYNCHRONIZE.RESPONSE action using using the SReSyncResponse routine.[2]

```
int     SReSyncResponse (sd, ssn, settings, data, cc, si)
int     sd;
int     settings;
long    ssn;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

sd: the session-descriptor;

ssn: the serial-number to resynchronize to;

settings: the new token settings;

data/cc: any additional data (and the length of that data, which may not exceed SN_SIZE octets); and,

si: a pointer to a SSAPindication structure, which is updated only if the call fails.

### 3.4.5 Activity Management

There are several types of activity management services: activity start and resume, activity interrupt and discard, and activity end.

#### Activity Start/Resume

To initiate a new activity, the SActStartRequest routine is used (which corresponds to the S-ACTIVITY-START.REQUEST action).

[2] Actually, the user has other choices by using the rules of contention resolution. Consult the ISO session service specification for the gruesome details.

```
int     SActStartRequest (sd, id, data, cc, si)
int     sd;
struct SSAPactid *id;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd`: the session-descriptor;

`id`: the activity-identifier;

`data/cc`: any additional data (and the length of that data, which may not exceed `SV_SIZE` octets); and,

`si`: a pointer to a `SSAPindication` structure, which is updated only if the call fails.

The `id` parameter is a pointer to a `SSAPactid` structure, which is passed transparently by the session service.

```
struct SSAPactid {
#define SID_DATA_SIZE   6
    u_char  sd_len;
    char    sd_data[SID_DATA_SIZE];
};
```

The elements of this structure are:

`sd_data/sd_len`: the data (and length of that data, which may not exceed `SID_DATA_SIZE` octets);

If the call to the `SActStartRequest` routine is successful, then the activity is started. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

To resume a previously interrupted activity, the `SActResumeRequest` routine is used (which corresponds to the S-ACTIVITY-RESUME.REQUEST action).

```
int     SActResumeRequest (sd, id, oid, ssn, ref, data, cc,
                           si)
int     sd;
struct SSAPactid *id,
                 *oid;
long    ssn;
struct SSAPref *ref;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor;

`id:` the activity-identifier (consult page 81 for a description of the `SSAPactid` structure);

`oid:` the previous activity-identifier (again, consult page 81);

`ssn:` the serial-number to resume the activity at;

`ref:` the previous connection identifier;

`data/cc:` any additional data (and the length of that data, which may not exceed `SV_SIZE` octets); and,

`si:` a pointer to a `SSAPindication` structure, which is updated only if the call fails.

The `ref` parameter is a pointer to a `SSAPref` structure. Note that unlike the connection identifiers used during connection establishment (as described on page 52), there are four fields:

| Field | Contents | Length |
|---|---|---|
| calling SSAP user reference | `sr_calling` | `sr_calling_len` |
| called SSAP user reference | `sr_called` | `sr_called_len` |
| common reference | `sr_cdata` | `sr_clen` |
| additional reference | `sr_adata` | `sr_alen` |

If the call to the `SActResumeRequest` routine is successful, then the activity is resumed. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

### Activity Interrupt/Discard

To interrupt an activity in progress, the `SActIntrRequest` routine is used (which corresponds to the S-ACTIVITY-INTERRUPT.REQUEST action).

```
int     SActIntrRequest (sd, reason, si)
int     sd;
int     reason;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor;

`reason:` the reason for the interrupt (codes are listed in Table 3.5); and,

`si:` a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the call to `SActIntrRequest` is successful, then the activity interrupt has been queued for the remote user. When the S-ACTIVITY-INTERRUPT.CONFIRMATION event is received, the activity interrupt is complete. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

Upon receiving a S-ACTIVITY-INTERRUPT.INDICATION event, the user is required to generate a S-ACTIVITY-INTERRUPT.RESPONSE action using the `SActIntrResponse` routine.

```
int     SActIntrResponse (sd, si)
int     sd;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor; and,

`si:` a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the call to `SActIntrResponse` is successful, then the activity interrupt has been completed. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

To discard an activity in progress, the `SActDiscRequest` routine is used (which corresponds to the S-ACTIVITY-DISCARD.REQUEST action).

```
int     SActDiscRequest (sd, reason, si)
int     sd;
int     reason;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor;

`reason:` the reason for the discard (codes are listed in Table 3.5); and,

`si:` a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the call to `SActDiscRequest` is successful, then the activity discard has been queued for the remote user. When the S-ACTIVITY-DISCARD.CONFIRMATION event is received, the activity discard is complete. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

Upon receiving a S-ACTIVITY-DISCARD.INDICATION event, the user is required to generate a S-ACTIVITY-DISCARD.RESPONSE action using the `SActDiscResponse` routine.

```
int     SActDiscResponse (sd, si)
int     sd;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor; and,

`si:` a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the call to `SActDiscResponse` is successful, then the activity discard has been completed. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

### Activity End

To end an activity in progress, the `SActEndRequest` routine is used (which corresponds to the S-ACTIVITY-END.REQUEST action).

```
int     SActEndRequest (sd, ssn, data, cc, si)
int     sd;
long   *ssn;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd`: the session-descriptor;

`ssn`: a pointer to a long integer which, on a successful return, will be updated to reflect the current serial-number ($V(M) - 1$);

`data/cc`: any additional data (and the length of that data, which may not exceed `SV_SIZE` octets); and,

`si`: a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the call to `SActEndRequest` is successful, then the activity end has been queued for the remote user. When the S-ACTIVITY-END.CONFIRMATION event is received, the activity end is complete. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

Upon receiving a S-ACTIVITY-END.INDICATION event, the user is required to call the `SActEndResponse` routine to generate a S-ACTIVITY-END.RESPONSE action.

```
int     SActEndResponse (sd, data, cc, si)
int     sd;
char   *data;
int     cc;
struct SSAPindication *si;
```

The parameters to this procedure are:

`sd`: the session-descriptor;

`data/cc`: any additional data (and the length of that data, which may not exceed `SV_SIZE` octets); and,

`si`: a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the call to `SActEndResponse` is successful, then the activity end has been completed. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

### 3.4.6 Exception Reporting

To report an exception and place the connection in a special error-recovery state, the `SUReportRequest` routine is called (which corresponds to the S-U-EXCEPTION-REPORT.REQUEST action).

```
int SUReportRequest (sd, reason, data, cc, si)
int sd;
int     reason;
char   *data;
int cc;
struct SSAPindication  *si;
```

The parameters to this procedure are:

`sd`: the session-descriptor;

`reason`: the reason for the report (codes are listed in Table 3.5);

`data/cc`: any report data (and the length of that data, which may not exceed `SP_SIZE` octets); and,

`si`: a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the call to `SUReportRequest` is successful, then the connection is placed in an error state, and any data queued for the connection may be lost until recovery is complete. Otherwise the `SSAPabort` structure contained in the `SSAPindication` parameter `si` contains the reason for failure.

Typically, error recovery can be achieved by giving away the data token or by aborting the connection (discussed next). Error recovery is discussed in greater detail in the ISO session service specification.

### 3.4.7 User-initiated Aborts

To unilaterally initiate an abort of the connection, the `SUAbortRequest` routine is called (which corresponds to the S-U-ABORT.REQUEST action).

```
int SUAbortRequest (sd, data, cc, si)
int sd;
char   *data;
int cc;
struct SSAPindication  *si;
```

The parameters to this procedure are:

`sd`: the session-descriptor;

`data/cc`: any abort data (and the length of that data, which may not exceed `SA_SIZE` octets); and,

`si`: a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the call to `SUAbortRequest` is successful, then the connection is immediately closed, and any data queued for the connection may be lost.

### 3.4.8 Asynchronous Event Handling

The data transfer events discussed thus far have been synchronous in nature. Some users of the session service may wish an asynchronous interface. The `SSetIndications` routine is used to change the service associated with a session-descriptor to or from an asynchronous interface.

```
int     SSetIndications (sd, data, tokens, sync, activity,
                report, finish, abort, si)
int     sd;
int   (*data) (),
      (*tokens) (),
```

```
        (*sync) (),
        (*activity) (),
        (*report) (),
        (*finish) (),
        (*abort) ();
 struct SSAPindication  *si;
```

The parameters to this procedure are:

`sd:` the session-descriptor;

`data:` the address of an event-handler routine to be invoked when data has arrived;

`tokens:` the address of an event-handler routine to be invoked when a token management event occurs;

`sync:` the address of an event-handler routine to be invoked when a synchronization management event occurs;

`activity:` the address of an event-handler routine to be invoked when an activity management event occurs;

`report:` the address of an event-handler routine to be invoked when an exception report event occurs;

`finish:` the address of an event-handler routine to be invoked when the connection is ready to be released;

`abort:` the address of an event-handler routine to be invoked when a user-initiated abort (a S-U-ABORT.INDICATION event occurs) or a provider-initiated abort (a S-P-ABORT.INDICATION event occurs); and,

`si:` a pointer to a `SSAPindication` structure, which is updated only if the call fails.

If the service is to be made asynchronous, then all event handlers are specified, otherwise, if the service is to be made synchronous, then no event handlers should be specified (use the manifest constant `NULLIFP`). The most