# OASIS SECURITY SERVICES DYNAMIC SESSION SPECIFICATION WORKING DRAFT

Last Updated: 05/10/01

SSTC filename: draft-sstc-sessions-dynamic-00.doc
[Submitter's version of original submission: 0.51]

Editor(s):
David Orchard, Jamcracker
Gilbert Pilz, Jamcracker

Jamcracker, Inc.

# Abstract

This document provides a framework for specific interactions around session management to occur between a provider of a session and a consumer of a session. A typical use of this is for the synchronization of a single sign-on assertion. It addresses the needs of single sign-off, single time-out, and single maintain session. This is termed dynamic sessions.


# Status of this Document

This document is a Working Draft, issued by the OASIS security services session subgroup. It is intended to be a standalone document that layers on the OASIS security services specification.

The team expects that significant changes will occur in this document before version 1.0 is released. The Team will not allow early implementation to constrain its ability to make changes to this specification prior to final release.

# Relationship to other standards

## Normative

XML:
http://www.w3.org/TR/REC-xml

SOAP:
http://www.w3.org/TR/SOAP/

XML Schema

SAML

# Audience

This document is a technical specification and is intended for developers and architects.

# Document Conventions

The following notations are used to present material in this document:

**ISSUE:** An issue is a direct request for feedback from the audience. An issue reflects a lack of decision due to insufficient or conflicting inputs. These are resolved through the acquisition of more input

**NOTE:** Extra normative information that the author(s) wish to draw the attention of the reader to.

# Use Cases, Scenarios and Requirements

The following Use cases and scenarios describe the interactions supported by this specification.

## A. Browser-based Single Sign-on to ecosystem

Note that this is a duplicate of Oasis security Services Scenario #1

1. A user logs on to the authority web site.
2. The accesses a recipient site that participates in the session management ecosystem. The user is not prompted for an authentication credential. Authorization to resources is controlled by the Session Recipient. This use case may be described as centralized authentication and delegated authorization.

## B. Time-Out

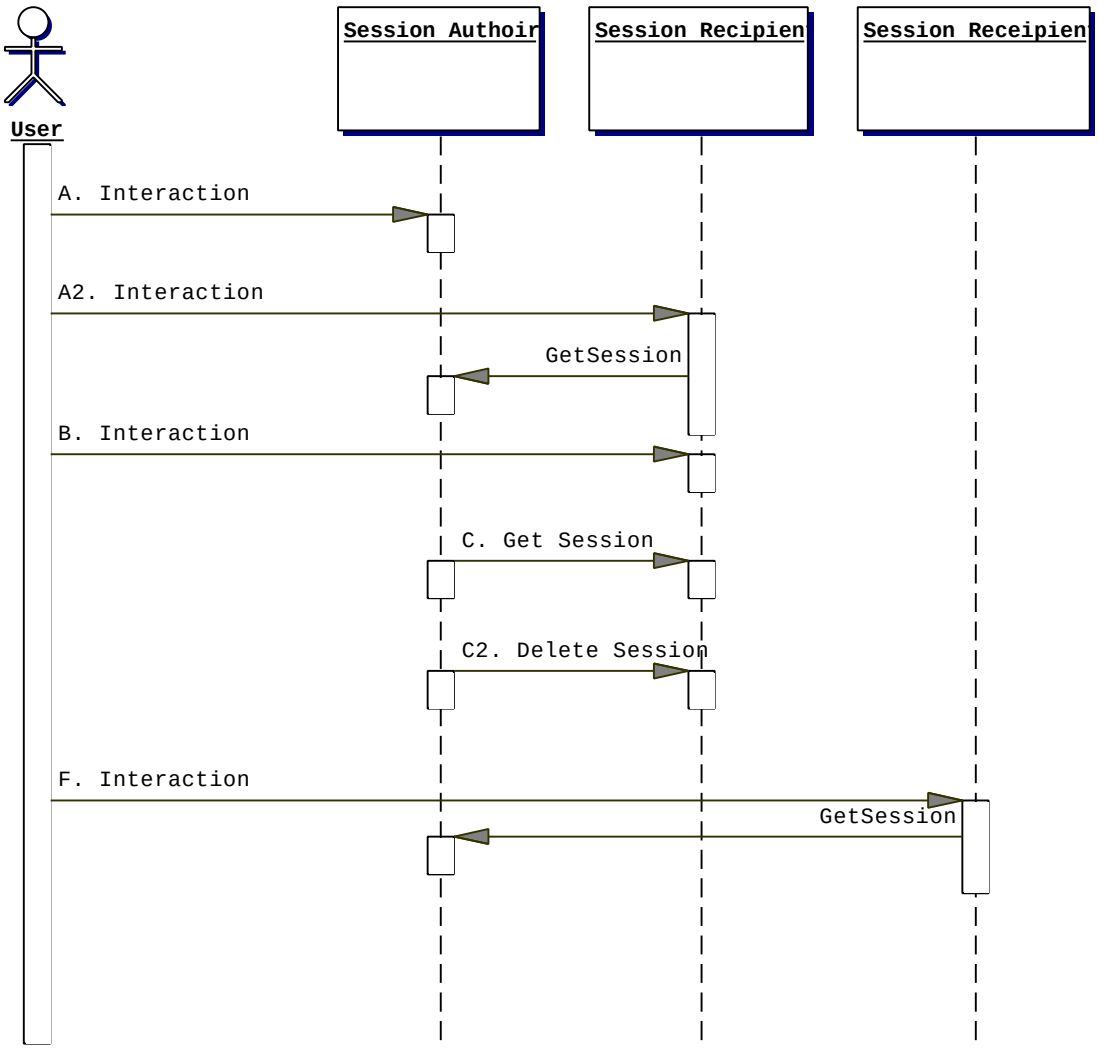
1. A user has abandoned their interactions with the ecosystem. The session must be purged from all machines participating in the session.

## C. Log-off

A user has signed on to the authority web site. They can choose to log out of the entire ecosystem by logging out of the authority web site.

## Session Management Scenarios

The following diagram illustrates the sign-on, time-out and log-out session management scenarios.




. A Single Sign-on and hand-off

Note that this is a duplicate of Oasis security Services Scenario #1

1. A user logs onto the authority Web site. This results in the creation of a session on the authority web site.
2. User requests a link to a recipient web site. This link contains an authentication reference/token/ticket.
3. User requests resource represented by link on recipient web site, including reference
4. Recipient web site requests validation of authentication reference from authority web site.
5. Authority web site returns success or failure, optionally additional session information.
6. Recipient web site returns web site to user

B. Timeout

1. Assume that the user has gone beyond the timeout limit on the authority web site.
2. The authority web site will query each participating web site to determine if the user has been active on their web site.
3. If the user has not been active on any of the recipient web sites within the timeout period, the recipient web sites are instructed to delete the session.

C. Logout

1. User logs out of the authority web site.
2. Each of the recipient web sites are instructed to delete the session.

**Notes:**

From the users perspective, the logout and timeout happen as if there was no session management. When they press logout, the Session Recipient controls the screen. The performance impact of refreshing master and slave sessions is not seen by any end-user.

## Functional and non-functional Requirements

1. The session information shall map easily to web sessions.
2. The algorithm must be robust in the case of failure of a communication path or node
3. The algorithm must allow for Session Recipients to not notify in the case of update to a local session.
4. The algorithm must allow for failure of Session Recipients
5. The algorithm must work with version 4.0 + web browsers and wireless devices
6. A local session can be refreshed from a global session. Alternate definition: A user timed-out from a Session Recipient should be able to re-access the Session Recipient based on valid sessions at the session authority.
7. The session recipient must invalidate a user's session assertions upon sign-off. **ISSUE [Other assertions]:** Should other assertions be invalidated? Probably dependent upon design.
8. User Sign-off at a session recipient is supported.
9. Session recipients shall be able to control timeout independently of session authority. **Issue: [Timeout differences]:** Assuming authority and recipient timeouts are supported, what is the algorithm for determining the current timeout. Example A. Time out on authority is 15 minutes. Timeout on recipient is 10 minutes. User has accessed authority and recipient 12 minutes ago. Should timeout of recipient occur (>10 minutes) or not (<15 minutes for authority). Example B. Timeout on authority is 10 minutes, timeout on recipient is 15 minutes, user last accessed both 12 minutes. Same question.
   Editor recommends: timeout master is the authority web site, and recipient web sites can shorten for their sites. In case A, user times out of recipient but not authority. In case B, user should have been deleted from recipient site because the authority will have sent a delete session message.

## Non Requirements

1. Access to a Session Recipient before access to Session Authority is not required. That is, there is no requirement for complex redirecting from Session Recipient to Session Authority back to Session Recipient if correctly logged on. Microsoft Passport, Netegrity Siteminder, etc. provide this functionality and we are loath to re-create it.
2. Offline support is not required.
3. It is required that Session Authority be informed of accesses to Session Recipient systems for the purposes of billing, anonymity is a bug not a feature.
4. The session recipient should not invalidate a user's assertions upon time-out.
5. The session authority shall not require a session recipient to time-out. **ISSUE[duplicate timeout req]:** Is this a duplicate of req #9, just from the authority perspective?

## Assumptions

A primary assumption has been made:

*1. "Users tend to interact frequently within an Session Recipient and infrequently interact across multiple Session Recipients".*

This assumption allows the use of a conversation between Session Authority and a Session Recipient when for purposes of timing out and logging out.

1. There is a prior existing trust relationship between Session Authority and the Session Recipient.
2. The Content of the session is defined separately, such as SAML.
3. Authorization information exchange is not required.
4. A session recipient will only have to deal with one logical session authority for a given user for a given session.

## Design Goals

1. Timeliness of assertion updates is preferred over efficiency of interactions.

## Glossary

**Dynamic Session –**

**Global Session –** The collection of session assertions residing at a session authority, containing the session authority's view of session across all recipients.

**Local Session –** The session assertion at the session recipient.

**Session Assertion** – The data of the session

**Session Authority** – The originator of the session.

**Session Recipient** – An application provided over the net and typically charged by the month. Note that there is no technical difference between a Session Recipient and a web site. An Session Recipient can provide a single or multiple business processes. **ISSUE:** Should this be relying party?

**Sign-off** – A user indicates to the session authority that all active assertions are no longer valid.

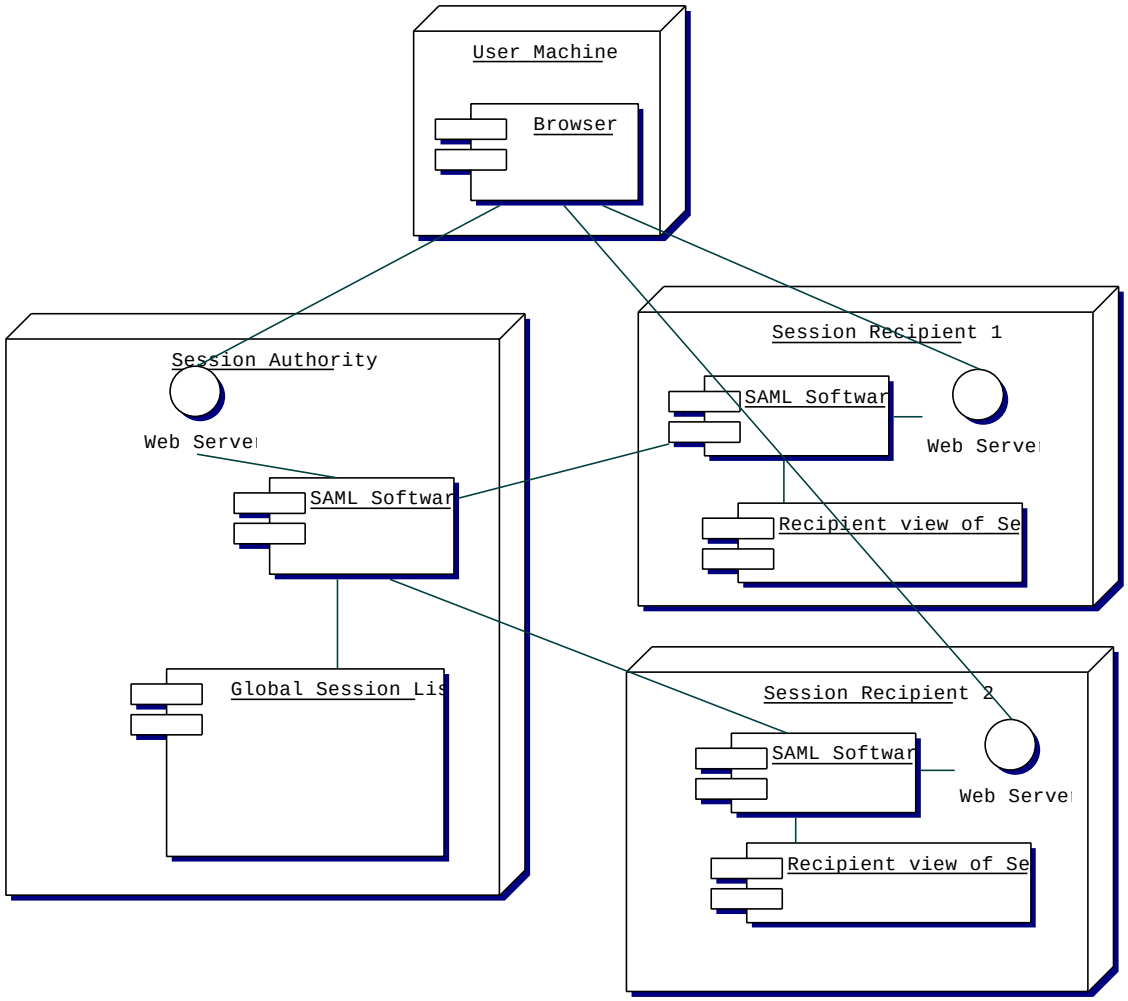
**Sign-on –** A user presents credentials to an authentication assertion. The authentication authority informs a session authority that sessions may be requested by session recipients.

**Time-out** – A user has not interacted with a system within a period of time. This generally indicates that the users active assertions should become invalid

# Session Management architecture

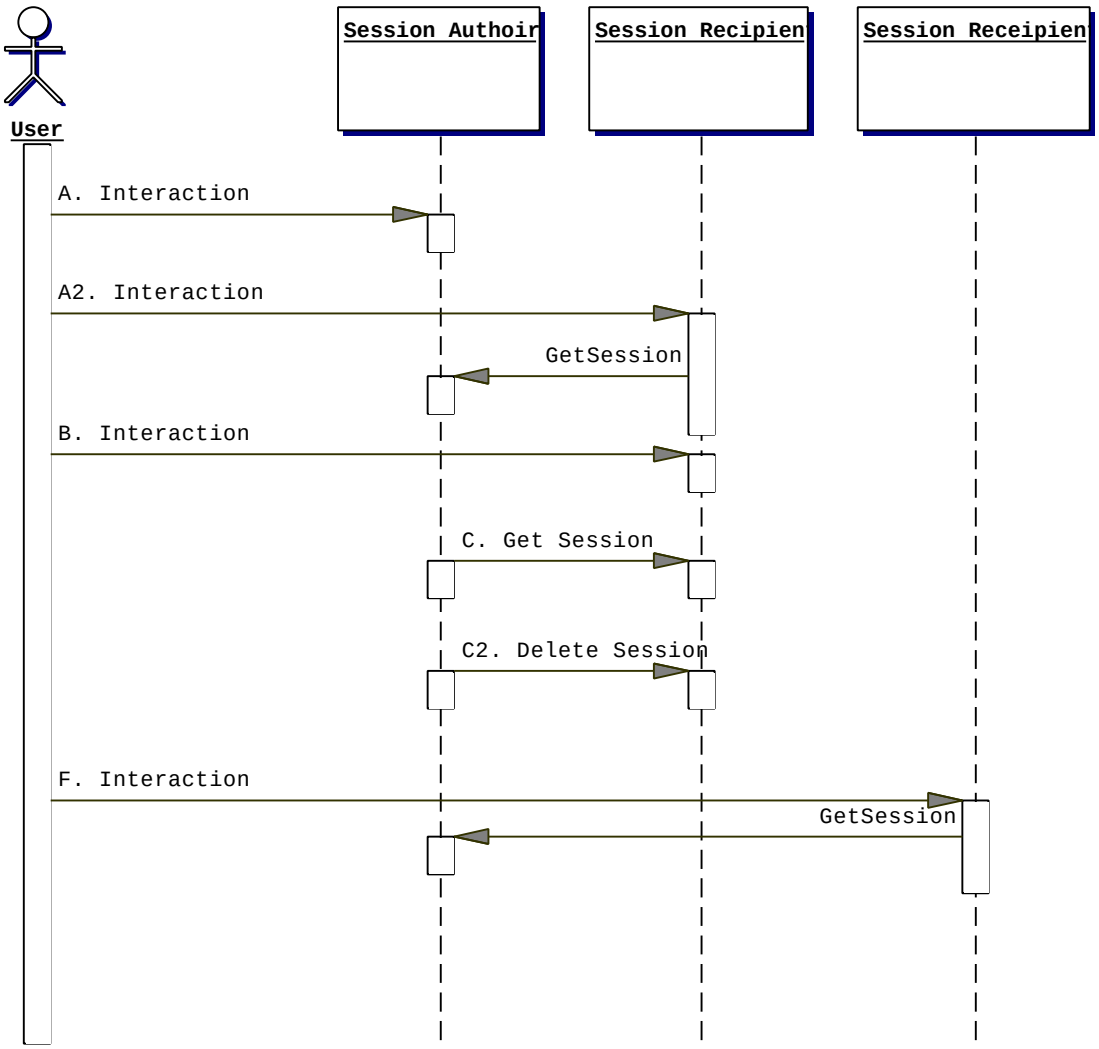
## Deployment diagram

The following diagram shows the deployment of the session management:




## Session Management interaction diagrams

The session management interactions are shown below




The use of Single sign-on being transferred by session management should help.

A. Simple logon and hand-off

7. A user logs onto Session Authority. This results in the creation of a session on Session Authority
8. The user then accesses Session Recipient1. This results in Session Recipient1 creating a session and then requesting the session from Session Authority via a “getSession” message.
9. Session Authority adds the Session Recipient to the list of active Session Recipients for the given user
10. Session Authority sends an “addSession” response
11. The user is allowed access to Session Recipient1

B. Session Recipient Interaction

1. User accesses Session Recipient1 again. There is no interaction between Session Recipient and Session Authority.

C. Session Authority Timeout

4. Assume that the user has gone beyond the timeout limit on Session Authority. Session Authority will check the active Session Recipient session that it has attached to this

user's Session Authority session. Session Authority will send a "getSession" message to every Session Recipient to determine if the user has an active session at this time.

5. If no active sessions or user has selected Logout, Session Authority will delete the session and notify every Session Recipient active for the user session via "deleteSession" message.

D. Logout

3. User logs out of

D. Session Recipient Timeout

1. Assume that the user has gone beyond the timeout limit on Session Recipient. The Session Recipient logs the user out without checking with the rest of the ecosystem. Should the user request access to the Session Recipient again, the Session Recipient can simply follow step A2 which does not prompt the user for any information. There is no communication from Session Recipient to Session Authority.

E. Session Recipient Logout

1. User requests logout from Session Recipient. There is no communication from Session Recipient to Session Authority.

F. Session Recipient1 to Session Recipient2 hand-off

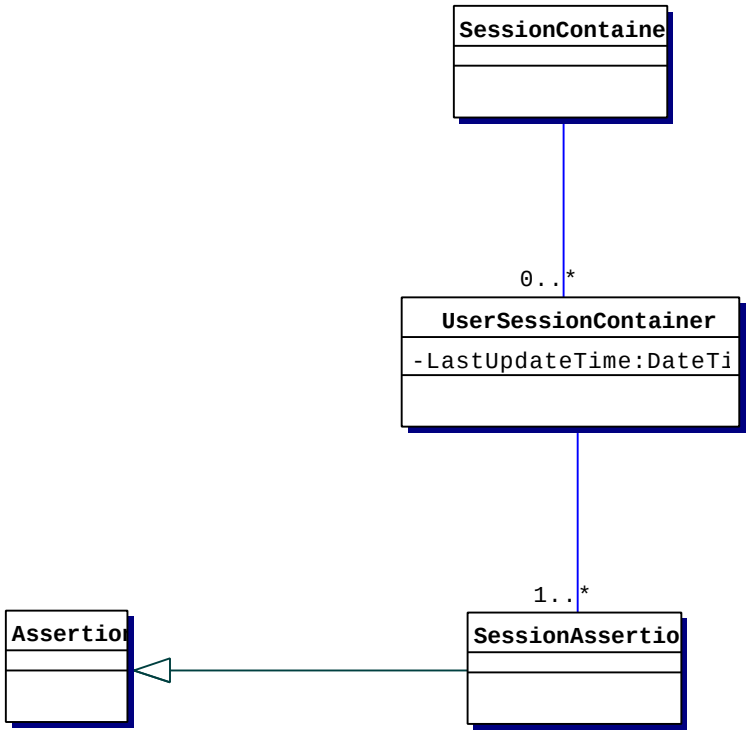
1. Starting at step A2, the user then accesses Session Recipient2
2. Session Recipient2 queries Session Authority for the session.
3. Session Authority returns the session to Session Recipient2. There is no update to Session Authority

Notes:

From the users perspective, the logout and timeout happen as if there was no session management. When they press logout, the Session Recipient controls the screen. The performance impact of refreshing master and slave sessions is not seen by any end-user.

## Dynamic Session Class Diagram

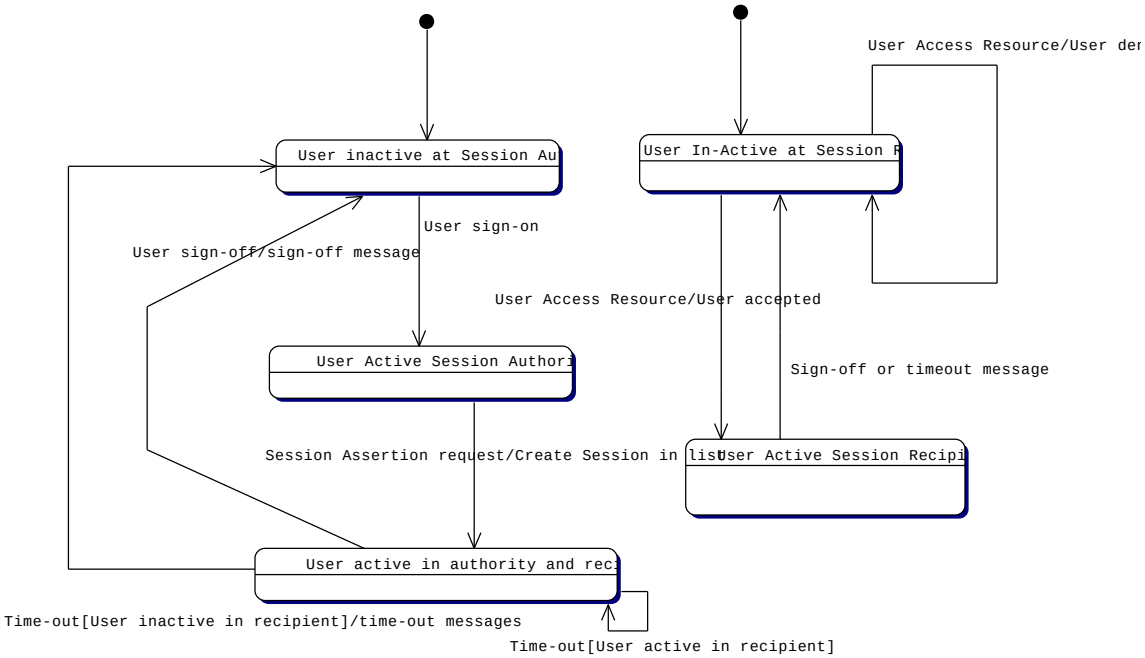
The class diagram for Dynamic Session follows:




The session container has a collection of user Session Containers. Each Container can have many sub-elements.

## Dynamic Session State Transition Diagram

The class diagram for State Transition Session follows:



**NOTE:** This diagram needs considerable work.  It is unclear what the states and transitions should be.

# Design guidelines

This specification describes the following key decisions:

- The session authority pulls state changes from Session Recipients
- Session Authority can pull from many session recipients.
- Times of actions on state changes are deltas from last pull, not absolute
- The entire session for a given user is sent for each request

The assumption of lengthy interactions with a single Session Recipient at a time allows the use of a Session Authority centric polling model. Session Authority pulls state changes from Session Recipients for the purposes of checking timeouts. Session Recipients push state changes when a logout occurs.

In total, there are 6 possible options for synchronization.

1. Session Authority pull state whenever it needs (typically because of user request on Session Authority or Session Recipient). This is the design selected for timeout
2. Session Authority pulls state at a periodic interval
3. Session Recipient pushes state to Session Authority whenever a change happens. This is selected for log-out.
4. Session Recipient pushes state at a periodic interval.
5. Session Recipient pulls state from Session Authority whenever a potential change could occur (timeout)
6. Session Recipient pulls state at a periodic interval

The key advantage of the Session Authority pull approaches is that Session Authority is not dependent upon the Session Recipient for state management. If the Session Recipient is down then Session Authority will deal with the Session Recipient being down, as opposed to not receiving an update. The key advantage for the “whenever” needed approach is that the session will be more timely – that is Session Authority will pull the data when it needs so it will be current.

The downside of the Session Authority pull approach is that there may be a significant performance hit when Session Authority pulls the data from the Session Recipient. This approach may also result in significantly increased traffic between Session Authority and the Session Recipient. There may be an interaction between Session Authority and the Session Recipient for every user on the Session Authority ecosystem.

Session Recipients may push state to Session Authority only when the session is being deleted from their system. Other updates to the state on their system are propagated only when Session Authority pulls the data.

The time associated with each change to a session is a delta time, with time zero being the time of the last update from Session Authority. This prevents the problem of systems having different values for the current time.

The changes propagated by the session management are the current value. The alternative is deltas or log of updates to the store. Thus add/delete/modify on various elements within the session are transmitted.

The entire session for a given user is always transmitted. It is anticipated that the data set will be fairly small (<5 Kilobytes) so optimization of transmitting subgraphs is not necessary

SessionIDs are assigned by Session Authority. Session Authority assures that the sessionIDs are unique across the space of Session Recipients it deals with.

**Issue [ Pipeline ]** Should Session Authority combine multiple requests into a single request? For example, every request within a 1 second period is combined so there is only 1 message/second flowing.
**Issue [ update conflict algorithm]**: Is there an algorithm for update conflicts required?
**Issue [ update conflict notification]:** If there is an update conflict algorithm to be used on the session recipient, should the result be sent back to Session Authority? Ie, update(sessionID, session) returns a new session?
**ISSUE [ delete or keep session]:** Should session store a state (like deleted) instead of deleting the session? Or is requiring that Session Recipients update Session Authority on state changes (add/delete) sufficient?

# Session Management Messages

## Namespace

All Provisioning elements are associated with the sess namespace identifier, which is bound to the namespace URI http://www.itml.org/ns/2001/01/sessmgmt.

## Requests

The session management specification consists of a number of commands and responses.

Session Authority can issue the following requests to an Session Recipient with parameters are provided:

- GetSession(SessionID) – retrieve session
- DeleteSession(SessionID) – Render user session inactive.

An Session Recipient can issue the following requests to Session Authority

- GetSession(UserID) – retrieve a session
- DeleteSession(SessionID).

**NOTE:** There is no accepted standard for defining interfaces to XML/Web services, though WSDL appears to be gaining consensus. It is expected that this specification will evolve to any standard that emerges.

## Errors

The following error conditions are defined in the sessmgmt error namespace:

- InvalidSessionID
- InvalidSessionInfo
- InvalidCompanyID
- InvalidUserID

## Samples

The following show fragments of sample requests and response. These are not integrated with SOAP due to current authoring tool limitations on mixing namespaces.

## Sample Request - getSession

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by David Orchard (Jamcracker) -->
<!-- Instance of a fully formed getSession  -->
<sess:getSession xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns="http://www.itml.org/ns/2001/01/sessmgmt" xsi:noNamespaceSchemaLocation="ITMLSessMethods.xsd"
txid="abc:88:88:88:88">
    <sess:UserIdentity>
        <sess:UserID>dorchard</sess:UserID>
        <sess:CompanyID>Partner1</sess:CompanyID>
    </sess:UserIdentity>
</sess:getSession>
```

## Sample Response

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by David Orchard (Jamcracker) -->
<!-- Instance of a fully formed getSessionResponse  -->
<sess:getSessionResponse xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns="http://www.itml.org/ns/2001/01/sessmgmt"
xsi:noNamespaceSchemaLocation="ITMLSessMethods.xsd"
txid="abc:88:88:88:88">
     <sess:UserSessionContainer>
     <lastUpdateTime>+05</lastUpdateTime>
     <SessionIdentity>alargehardtoguessstring</SessionIdentity>

     <s2ml:NameAssertion xmlns="http://www.s2ml.org">
      <This>urn:authEngine32:xsde12</This>
      <Issuer>http://www.jamcracker.com</Issuer>
        <Date>2000-10—16T12:34:120-05:00</Date>
        <Audiences>urn:jceecosystem</Audiences>
          <AuthData>
        <AuthType>Login</AuthType>
       <IdentityToken>x12+21defqa$3#</IdentityToken>
      </AuthData>
      <dsig:signature>. . . </dsig:signature>
     </s2ml:NameAssertion>

     <bpi:bpdata xmlns="http://www.itml.org/ns/2001/05/bpi">
     </bpi:bpdata>
     </sess:UserSessionContainer>
</sess:getSessionResponse>
```

## Sample Response, error

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v3.5 NT (http://www.xmlspy.com) by David Orchard (Jamcracker) -->
<!-- Instance of a fully formed getSessionResponse  -->
<sess:getSessionResponse xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xmlns="http://www.itml.org/ns/2001/01/sessmgmt" xsi:noNamespaceSchemaLocation="ITMLSessMethods.xsd"
txid="abc:88:88:88:88">
     <sess:ITMLFaultDetail>
          <sess:faultcode>InvalidUserID</sess:faultcode>
          <sess:faultstring>C'est une nomme invalide</sess:faultstring>
     </sess:ITMLFaultDetail>
</sess:getSessionResponse>
```

# Appendix

## Session Message Schemas

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" elementFormDefault="qualified">
    <xsd:include schemaLocation="ITMLSessStructs.xsd"/>
    <xsd:element name="getSession">
        <xsd:complexType>
            <xsd:choice>
                    <xsd:element name="UserIdentity" type="UserIdentityType"/>
                    <xsd:element name="SessionIdentity" type="SessionIdentityType"/>

            </xsd:choice>
            <xsd:attribute name="txid" type="txidType"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="getSessionResponse">
        <xsd:complexType>
            <xsd:choice>
                <xsd:sequence>
                    <xsd:element name="UserSessionContainer" type="UserSessionType"/>
                </xsd:sequence>
                <xsd:element name="ITMLFaultDetail" type="ITMLFaultDetailType" minOccurs="0"/>
            </xsd:choice>
        <xsd:attribute name="txid" type="txidType"/>

        </xsd:complexType>
    </xsd:element>

        <xsd:element name="deleteSession">
        <xsd:complexType>
            <xsd:sequence>
                    <xsd:element name="UserIdentity" type="UserIdentityType"/>
                </xsd:sequence>
            <xsd:attribute name="txid" type="txidType"/>
        </xsd:complexType>
    </xsd:element>
    <xsd:element name="deleteSessionResponse">
        <xsd:complexType>
            <xsd:sequence>
            <xsd:element name="ITMLFaultDetail" type="ITMLFaultDetailType" minOccurs="0"/>
            </xsd:sequence>
        <xsd:attribute name="txid" type="txidType"/>

        </xsd:complexType>
    </xsd:element>
</xsd:schema>
```

## Session Structure Schemas

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Edited by David Orchard, Jamcracker -->
<xsd:schema xmlns:xsd="http://www.w3.org/2000/10/XMLSchema" elementFormDefault="qualified">
    <xsd:simpleType name="UserIDType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="CompanyIDType">
```

```
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:simpleType name="SessionIdentityType">
        <xsd:restriction base="xsd:string"/>
    </xsd:simpleType>
    <xsd:complexType name="UserIdentityType">
        <xsd:sequence>
            <xsd:element name="UserID" type="UserIDType"/>
            <xsd:element name="CompanyID" type="CompanyIDType"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="UserSessionContainer">
        <xsd:sequence>
            <xsd:element name="LastUpdateTime" type="xsd:timeDuration"/>
            <xsd:element name="SessionID" type="SessionIdentityType"/>
            <xsd:element name="UserSession" type="UserSessionType" maxOccurs="unbounded"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="UserSessionType">
        <xsd:sequence>
            <xsd:any namespace="##any" processContents="lax"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:complexType name="ITMLFaultDetailType">
        <xsd:sequence>
            <xsd:element name="faultcode" type="faultcodeType"/>
            <xsd:element name="faultstring" type="xsd:string"/>
        </xsd:sequence>
    </xsd:complexType>
    <xsd:simpleType name="faultcodeType">
        <xsd:restriction base="xsd:string">
            <xsd:enumeration value="InvalidUserID"/>
            <xsd:enumeration value="InvalidSessionID"/>
            <xsd:enumeration value="InvalidCompanyID"/>
            <xsd:enumeration value="InvalidSessionInfo"/>
        </xsd:restriction>
    </xsd:simpleType>
    <xsd:simpleType name="txidType">
        <xsd:restriction base="xsd:string">
            <xsd:pattern value="[a-z]{3}:[0-9]{2}:[0-9]{2}:[0-9]{2}:[0-9]{2}"/>
        </xsd:restriction>
    </xsd:simpleType>
</xsd:schema>
```

# Futures

- 

# Important Design notes (non-normative)

## Error Handling and Recovery

The specification has avoided any mention of error recovery or increased reliability. Typical examples of these are retrying the transaction, adopting a two-phase commit protocol, defining compensating transactions. Therefore it is likely that this will be a trouble spot for integration.

## Credits

This specification has been greatly helped by the following people and affiliations:

## Revision History

2000:

Dec 8 Initial revision
Dec 12: Version 0.2, Added schema, added design notes, added messages

2001:
Jan 1: Version 0.3. Conch model – there can be only one! Active Session Recipient at a time
Jan 15: Version 0.4. Conch model thrown out, cleaned up for publication.
Jan 30: Version 0.5. Cleaned up use cases, updated to latest XML Schema and wildcard
May 7: Version 0.51. Changed to conform to SAML guidelines