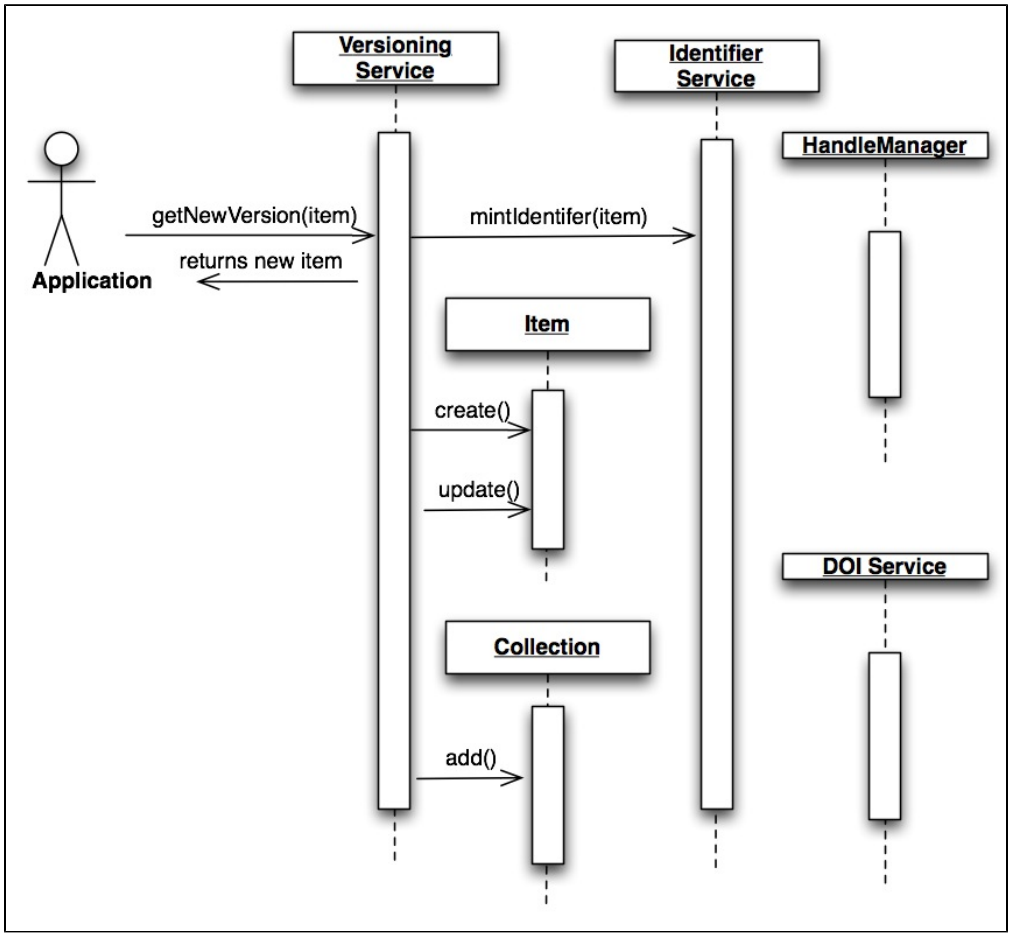
Services to support Alternative Identifiers

Together with the Item Level Versioning an Identifier Service was introduced that make it possible to integrate new Identifiers. Currently the Identifier Service is used for Items only, but this may be changed in future versions of DSpace. Identifiers used for different versions are an very important point as part of an versioning strategy. The following documentation describes the Identifier Service in the context of Item Level Versioning, nevertheless the Identifier Service is also used for Items when the Item Level Versioning is switched off.

Versioning and Identifier Service

DSpace Item Versioning is encapsulated as an Extensible Service that may be reimplemented by the local repository maintainers to produce alternate versioning behaviors and Identifier Schemes. Versioning Services layer on top of IdentifierServices dedicated to Encoding, Resolution, Minting and Registration of Identifiers for specific DSpace Items. It is through this highly extensible layering of functionality where local developers can alter the versioning behavior and introduce their own local enhancements. The DSpace Service Manager, based on the Spring Framework, provides the key leverage for this flexibility.



Versioning Service

The *Versioning* Service will be responsible for the replication of one or more Items when a new version is requested. The new version will not yet be preserved in the Repository, it will be preserved when the databases transactional window is completed, thus when errors arise in the *versioning* process, the database will be properly kept in its original state and the application will alert that an exception has occurred that is in need of correction.

The *Versioning* Service will rely on a generic *IdentifierService* that is described below for minting and registering any identifiers that are required to track the revision history of the Items.

```
public interface VersioningService {

    Version createNewVersion(Context c, int itemId);

    Version createNewVersion(Context c, int itemId, String summary);

    void removeVersion(Context c, int versionID);

    void removeVersion(Context c, Item item);

    Version getVersion(Context c, int versionID);

    Version restoreVersion(Context c, int versionID);

    Version restoreVersion(Context c, int versionID, String summary);

    VersionHistory findVersionHistory(Context c, int itemId);

    Version updateVersion(Context c, int itemId, String summary);

    Version getVersion(Context c, Item item);
}
```

Identifier Service

The Identifier Service maintains an extensible set of *IdentifierProvider* services that are responsible for two important activities in Identifier management:

1. Resolution: *IdentifierService* act in a manner similar to the existing HandleManager in DSpace, allowing for resolution of DSpace Items from provided identifiers.
2. Minting: Minting is the act of reserving and returning an identifier that may be used with a specific DSpaceObject.
3. Registering: Registering is the act of recording the existence of a minted identifier with an external persistent resolver service. These services may reside on the local machine (HandleManager) or exist as external services (PURL or EZID DOI registration services)

```
public interface IdentifierService {

    /**
     *
     * @param context
     * @param dso
     * @param identifier
     * @return
     */
    String lookup(Context context, DSpaceObject dso, Class<? extends Identifier> identifier);

    /**
     *
     * This will resolve a DSpaceObject based on a provided Identifier.  The Service will interrogate
the providers in
     * no particular order and return the first successful result discovered.  If no resolution is
successful,
     * the method will return null if no object is found.
     *
     * TODO: Verify null is returned.
     *
     * @param context
     * @param identifier
     * @return
     * @throws IdentifierNotFoundException
     * @throws IdentifierNotResolvableException
     */
    DSpaceObject resolve(Context context, String identifier) throws IdentifierNotFoundException,
IdentifierNotResolvableException;

    /**
     *
     * Reserves any identifiers necessary based on the capabilities of all providers in the service.
     *
     * @param context
     * @param dso
```

```
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void reserve(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
     *
     * Used to Reserve a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void reserve(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;

    /**
     *
     * @param context
     * @param dso
     * @return
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void register(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
     *
     * Used to Register a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @return
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void register(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;

    /**
     * Delete (Unbind) all identifiers registered for a specific DSpace item. Identifiers are "unbound"
across
     * all providers in no particular order.
     *
     * @param context
     * @param dso
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void delete(Context context, DSpaceObject dso) throws AuthorizeException, SQLException,
IdentifierException;

    /**
```

```
     * Used to Delete a Specific Identifier (for example a Handle,  hdl:1234.5/6) The provider is
responsible for
     * Detecting and Processing the appropriate identifier, all Providers are interrogated, multiple
providers
     * can process the same identifier.
     *
     * @param context
     * @param dso
     * @param identifier
     * @throws org.dspace.authorize.AuthorizeException
     * @throws java.sql.SQLException
     * @throws IdentifierException
     */
    void delete(Context context, DSpaceObject dso, String identifier) throws AuthorizeException,
SQLException, IdentifierException;

}
```