Overview

This use case extends the use case scenario from the 1.0 specification. In this scenario we expand on how OSLC services can helps a build Automation Plan govern and track ArtifactContribution to a AutomationResult. Though a software asset library has a wide range of use cases that are driven by user interfaces, this scenario is concentrated on an automated, mostly a headless OSLC interaction. The interaction of an asset management system with an orchestrated automation use case is a fundamental mean by which an asset library's content is kept compliant in terms of packaging (information, content, format, dependencies, related content, classification....), and the linking of these assets to the Bill of Materials (BOM), and overall life-cycle that is driving the creation of that asset. The Automation Plan that is driving this particular use case is a a build (Compile, Test, Scan...) process. But fundamentally, it interaction is similar to various kind of automation plans. Plans that drives a deploy (install, config, verification of a running instance) process, an aggregation process (collection and distilling of information from various sources into a homogeneous / consumable library content) etc.

Preconditions

This IT department has an OSLC enabled asset management system to govern their re-usable software assets (open source libraries, run-time binaries, compilers, models, documents, etc). The teams have an in house automated build system that runs their builds, deployments etc.

In order to automate the builds of several software development teams, there have been agreements made between the development and build teams to model the build environment and dependencies needed to build their product without having to involve a build engineer to make a change.

They have augmented their Automation plan to govern their build by interacting with an asset management system (user OSLC interfaces) to:

- Define the build configuration that drives the nightly build.
- Create a placeholder asset to be used as basis for a published implementation.
- Designate runtime and library dependencies used to build their software project.
- Publish the built implementations.

System Design

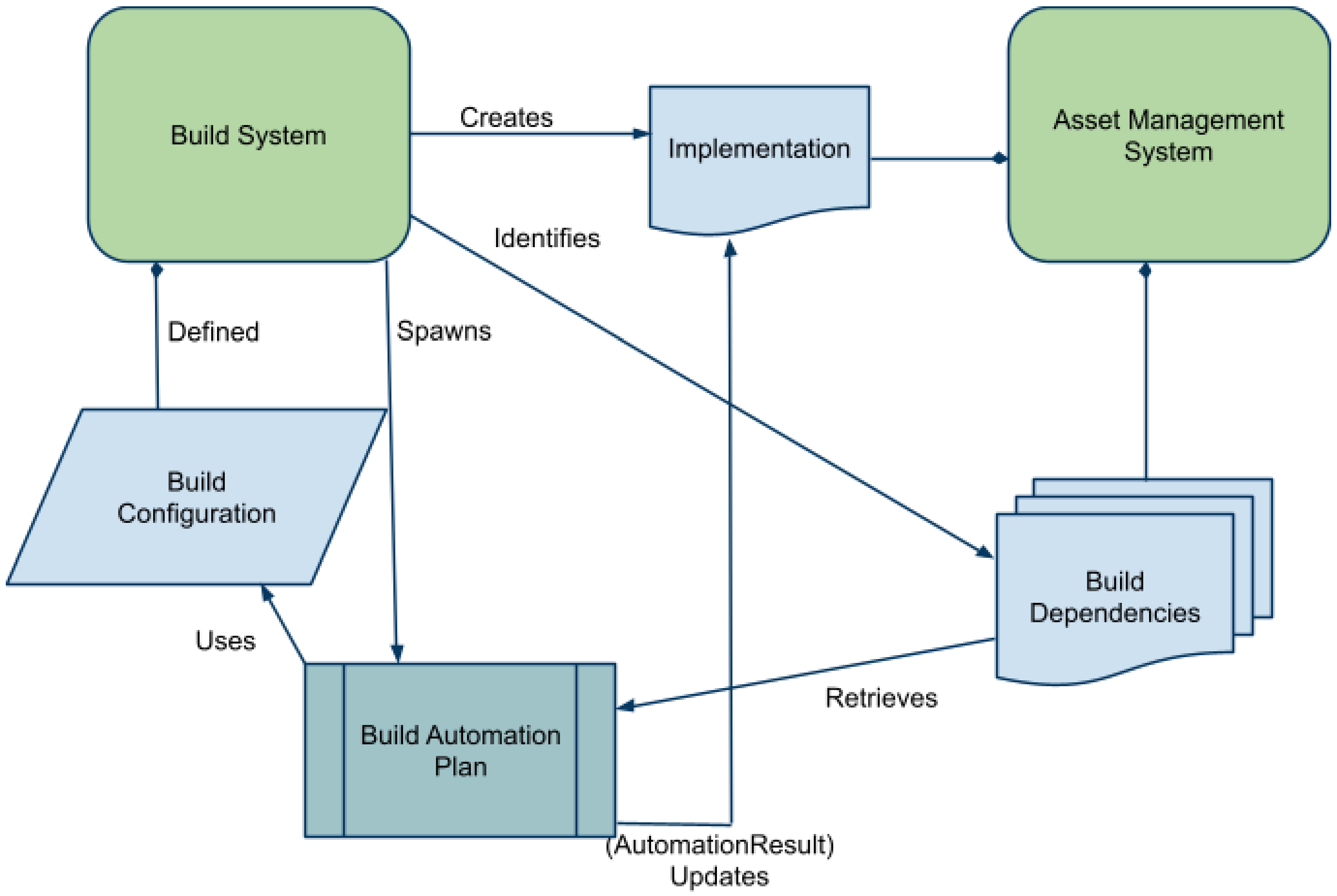


Figure 1 : Top level system design

As shown in Figure 1 there are two systems the users interact with that work together via OSLC services (the build system and the asset management system). Via the build system, the development team defines a build configuration that describes what needs to be done during a build of the development team's product. They also define an Implementation asset that works as a placeholder for the built work products.

During a build, the build process uses the build configuration to determine which resources and dependencies are needed. These resources could come from source control systems or published libraries in the asset management system. Once complete, the Build Process publishes the built asset to the asset management system.

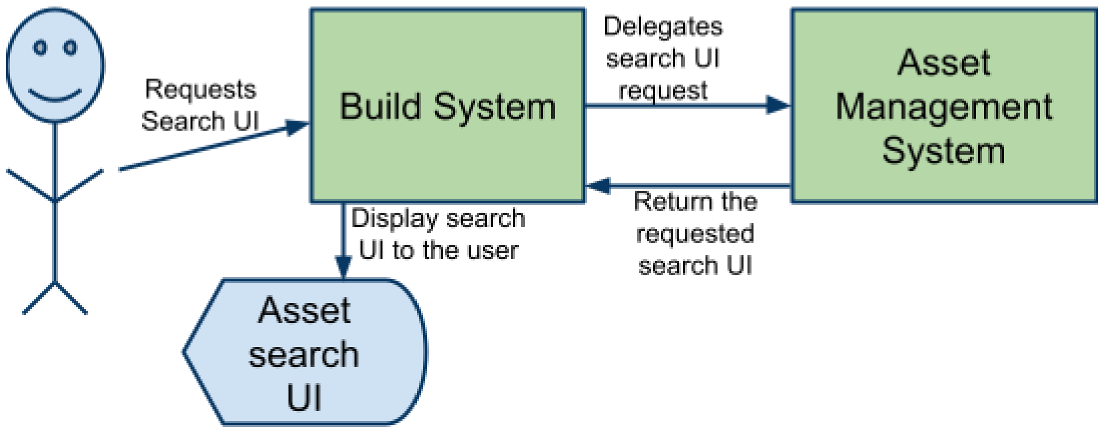
Scenario

The account services development team is building a web application to help better process account management requests. Their project's source code has been stored in a source control repository. The source depends on re-usable open source libraries stored in their asset management system.

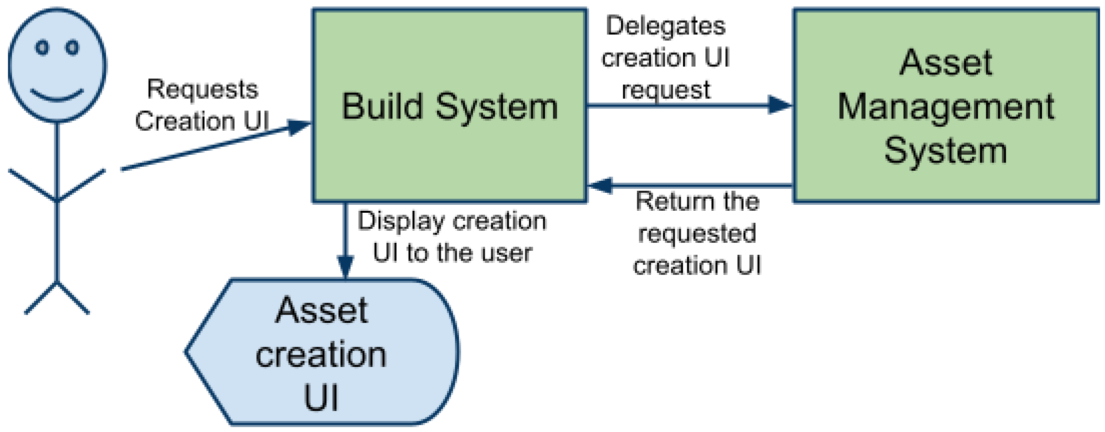
The project is at the point where they would like to schedule candidate automated builds. Through the build system interface they will define their build process. The build system will run this process nightly to build their implementation and publish the result back to the Asset Management system.

Defining the Build Configuration

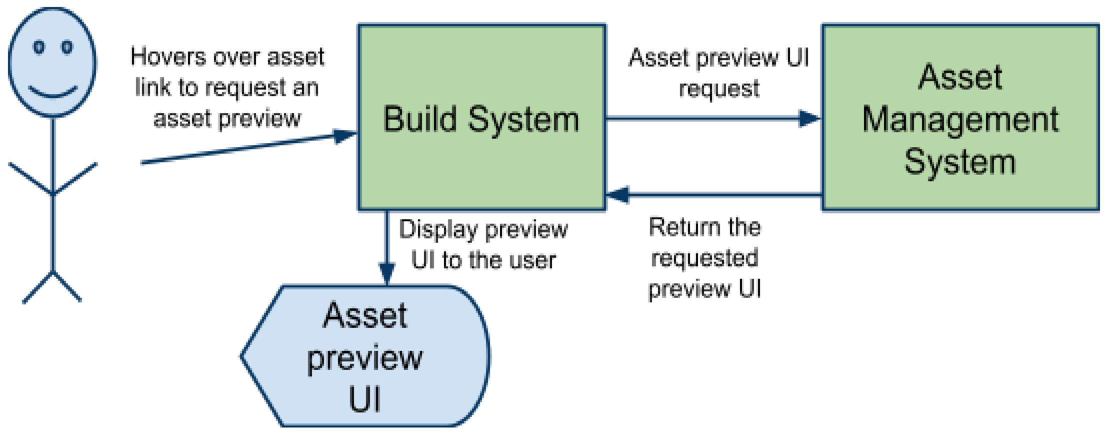
While defining the build configuration, the user opens a delegated UI within the build system UI that allows the build engineer to search within the Asset Management system for versions of dependent tools (compilers, libraries, code generators, etc) to be used during the build.


Requests Search UI
Build System
Delegates search UI request
Asset Management System
Return the requested search UI
Display search UI to the user
Asset search UI

The build engineer uses a delegated asset creation UI to create a placeholder asset (with links to the required tools found from the search) where the candidate build will be made available on publish.


Requests Creation UI
Build System
Delegates creation UI request
Asset Management System
Return the requested creation UI
Display creation UI to the user
Asset creation UI

When hovering over links in the build system UI, the build engineer is able to view a preview of the placeholder candidate build asset created.

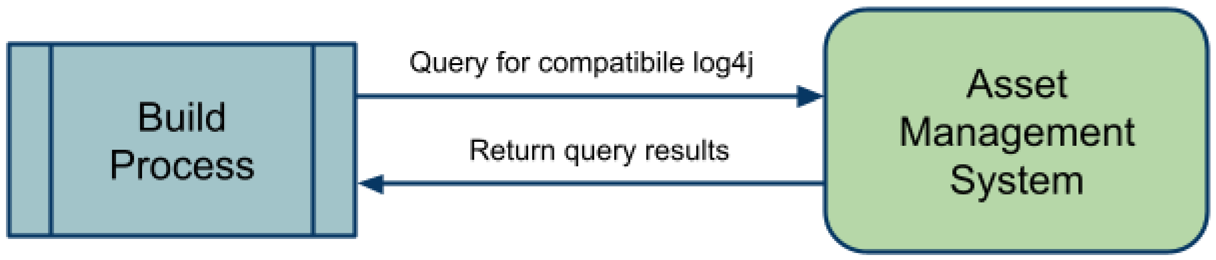

Hovers over asset link to request an asset preview
Build System
Asset preview UI request
Asset Management System
Return the requested preview UI
Display preview UI to the user
Asset preview UI

Build Process

During the candidate build, the configured build process uses Asset Management OSLC API to query, retrieve and publish assets from the asset management repository.

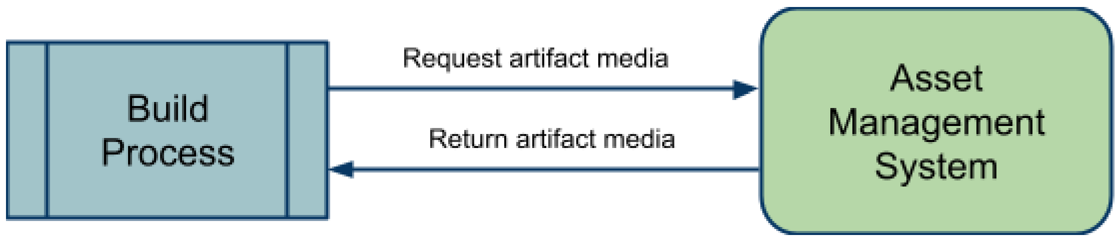
Query

System brings the source from SCM that includes information about the dependent bundle libraries (e.g. manifest, plugin.xml, maven POM files, build manifest etc). The build process discovers a dependency on the latest Approved bundles (for example Log4j with a version greater than or equal to 1.2 and less than 1.3) and uses the OSLC query capability to locate the most appropriate bundle.



Retrieve

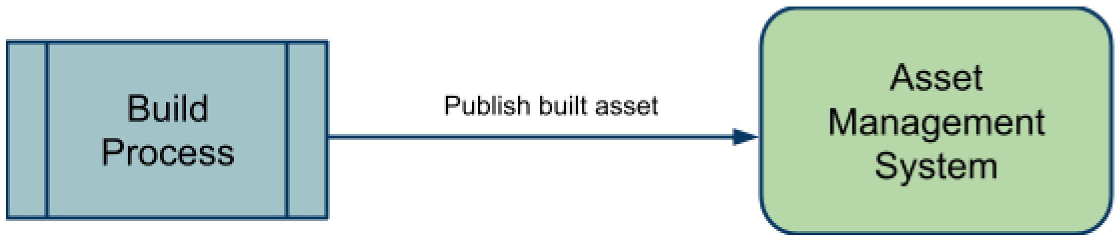
System downloads referenced library artifacts (tools from the the dependencies defined within the project) to be used during build.



Publish

The build system updates the placeholder asset with the built media resources.

Should we be adding references back to the OSLC automation resource?


Build Process
Publish built asset
Asset Management System