VERSION: 1.0

DATE: 3.7.2008

AUTHORS:

Thomas Brodkorb

SAP ARCHITECTURE BLUEPRINT

EDEN

End-2-end model driven Software Lifecycle Management

PROJECT NAME/CPROJECT TITLE: **NW712-L1:EDEN**

SPONSOR/PROJECT INITIATOR: **JUERGEN KREUZIGER**

PROGRAM/PROJECT LEAD: **THOMAS BRODKORB**

LEAD ARCHITECT: **THOMAS BRODKORB**

DEVELOPMENT: ☒ SAPLabs, mainly in WDF/SOF

☐ Partner/ISV _____

Document History		
Version	Date	Status (Comments)
19.6.2008	0.1	Start writing the document
30.6.2008	0.5	First draft version.
2.7.2008	0.9	First delivered version

I MARKET AND PRODUCT BACKGROUND OF PROJECT/PROGRAM

Planned release date:	NW 7.12	
Underlying SAP NetWeaver release:	NW 7.12	
Used SAP NetWeaver stacks:	☐ ABAP	☒ J2EE/Java EE 5

Use cases targeted by the project/program:

- Customer are supported in setting up their system landscapes by solution documentation and meta data
- Selections are made in terms the customer understands (e.g. capabilities – market terms)
- Meta data relate those selections to data that can be used to actually install and update the customer systems
- Technical system setup is automated (to a great extend) and (programmatically) driven by the selections the customer made in terms he understands
- In lower level – one system view
    - Capabilities (market term) translate to Functional Units (technical term)
    - Installers offer Functional Unit selections (if not automated from higher level tools like solution manager)
    - Functional Unit selections are translated to
    - installation units required to physically install the system (set of Software components)
    - Configuration units required to make the required Functional Units technically operable
    - A system then can be operated (technically) in terms of Functional Units

Strategic goals SAP wants to achieve with the project/program:

- Improve Software Lifecycle Management capabilities for our customers (lower TCO and TCU (Total Cost of Understanding))
    - Enable seamless integration of LM processes for the customer by using one common Meta model for all LM tools
- Enable the support of increasingly complex system landscapes
- Lower TCD by separating Software Lifecycle Management tools from content driving what the tools do for the customers
- Lower TCD by introducing on common Meta model

Mandatory software capabilities to address goals, use cases, and target market:

- …

…

II. ARCHITECTURE

End-2-End model driven Software Lifecycle Management - concept

The basic idea is to define one LM Meta Model which describes all artifacts that are relevant for Software Lifecycle Management. All LM tools then adapt this model in the sense that they interpret and adapt content of this LM Meta Model. The content of the LM Meta model could be called LM Meta data. However we prefer to call these data the LM model. Examples for entities in the LM Meta Model and LM Model respectively can be found in the following picture.

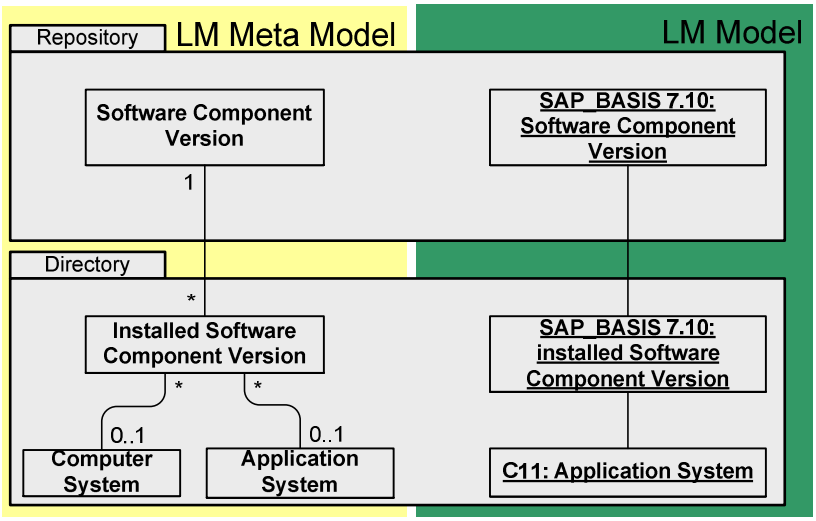

Figure 1: Examples of entities in the LM Meta model and LM model

Assuming such a LM Meta model exists we then define the Software Lifecycle Management processes operating on the data of this model (the LM model). We already stated that these processes shall cover everything from start to end. In previous versions of Software Lifecycle Management we consider the start point of those processes to be the moment the finished (assembled) software leaves SAP (or more general the producer of the software). Now we shift the start point of LM processes further to the front so that the development process is included. Obviously the development process always plays a major role in the software creation process. The new idea is to integrate the creation and maintenance of the LM model right into the development process. The software shall be self describing throughout the whole process.

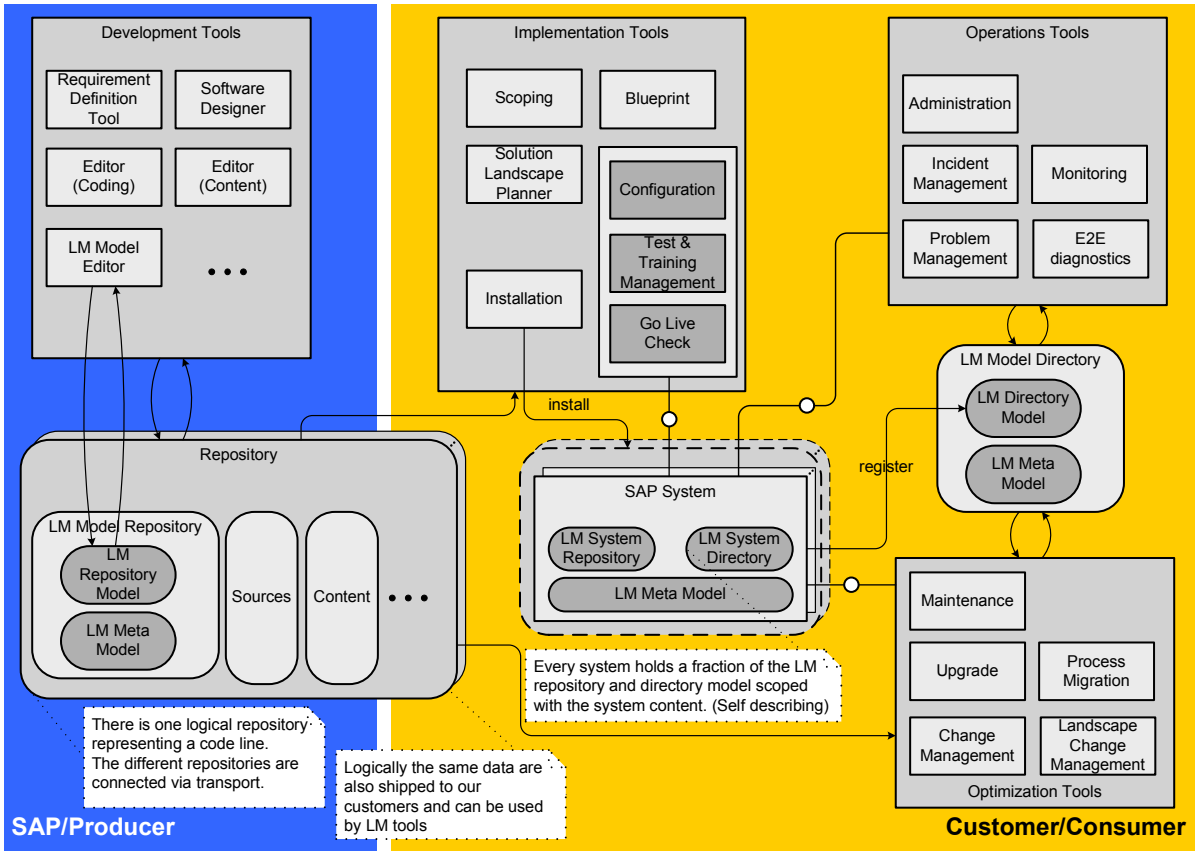

Figure 2: End-2-End model driven Software Lifecycle Management - static view

Figure 2 represents the components involved in the static view of such an end-2-end process creating and using data from the LM model. The central store for the LM Model is the “LM Model repository”. The process of creation data in this LM model shall be integrated in the development process at SAP or more generally speaking the development process of a software producer. Thus the “LM Model repository” is seen as only one part of the overall “repository” that holds all kinds of artifacts being subject to development. The components creating and maintaining these data are “Development Tools”. Of course not all development tools are described in this Blueprint. Development tools and repositories other than the “LM Model Editor” and the “LM Model repository” are shown to illustrate the context only.

The static view does not explicitly show the transport process which takes care of transferring the content of the repository (and artifacts generated from it) along software producer internal transport routes as well as from the producer to the consumer of the software.

Eventually the content of the repository being developed will be available at another instance of this repository running at the customer side. A set of “Implementation Tools” will then be able to operate on the data in the LM Model and guide the customer through the implementation process (Scoping, Blueprint, Solution landscape planner). At some point in time an “installation” will actually install a new system. (This is indicated by writing data to a store. The software finally ending up in that store (the FS of a host) will be started up and then become a (running) system.) Other tools (configuration, test & training management, go live check) then can operate on that system and complete the overall implementation process.

The concept of EDEN is that every installed system describes itself locally in terms of the LM Meta model. This is represented the three stores shown inside the system. From these data the system can actually report its state to the central LM Model directory.

Now “Operations Tools” can operate on data in the LM Model directory which offers details of the whole system landscape. Additionally introspection tools in the managed system (the component called system in the diagram) will deliver current data on the actual state of different components inside the management system. Those volatile data are not stored in the LM Model directory but directory transferred to the operations tools. Thus the LM Model directory holds more or less the static view of the system landscape only.

Finally “Optimization Tools” can compare eventually new or updated LM Model data with the current system states and support the customer in planning and executing update, upgrade or additional installation processes.

LM Meta Model

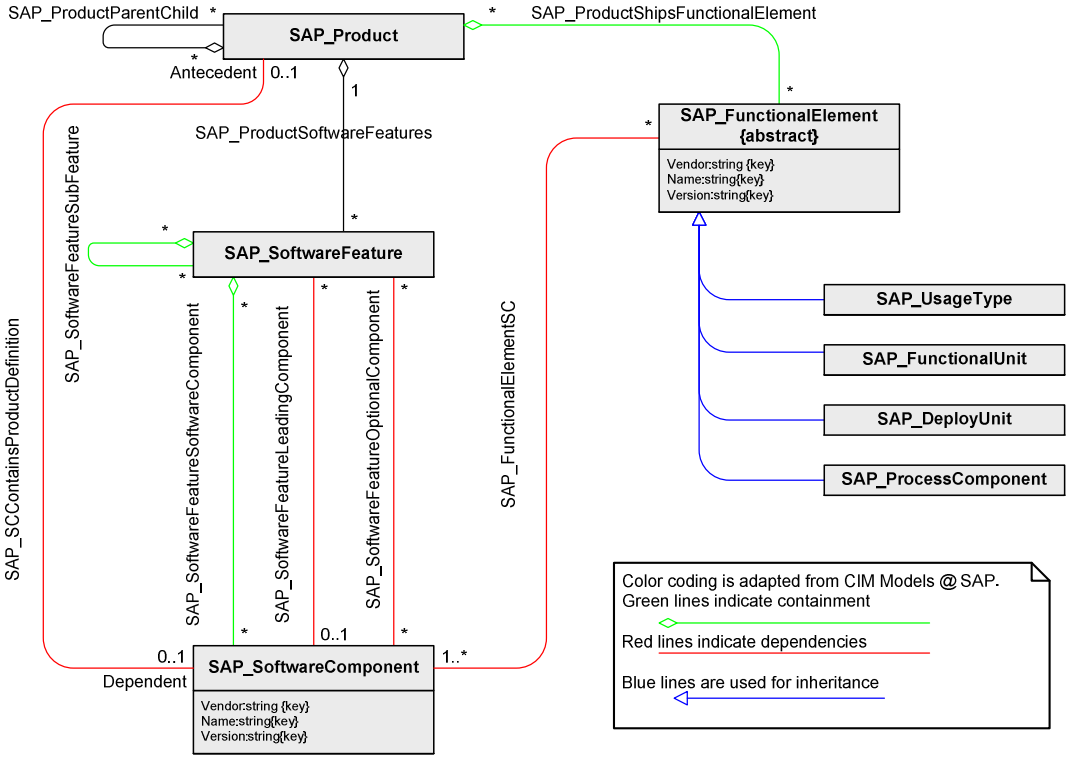

Figure 3: LM Meta Model Component Repository

The complete LM Meta Model is much more detailed. It is best viewed with the class browser in an SLD (System Landscape Directory) (e.g. http://slpmain.wdf.sap.corp:56700/sld (Anzeiger/Display) --> CIM Classes). Today most of the data modeled in the Component Repository part are originally stored in PPMS and synchronized into a central SLD at SAP. From there the data are shipped to the customer where they populate the LM Repository Model. From there the data are read and used by LM tools as shown above in Figure 2.

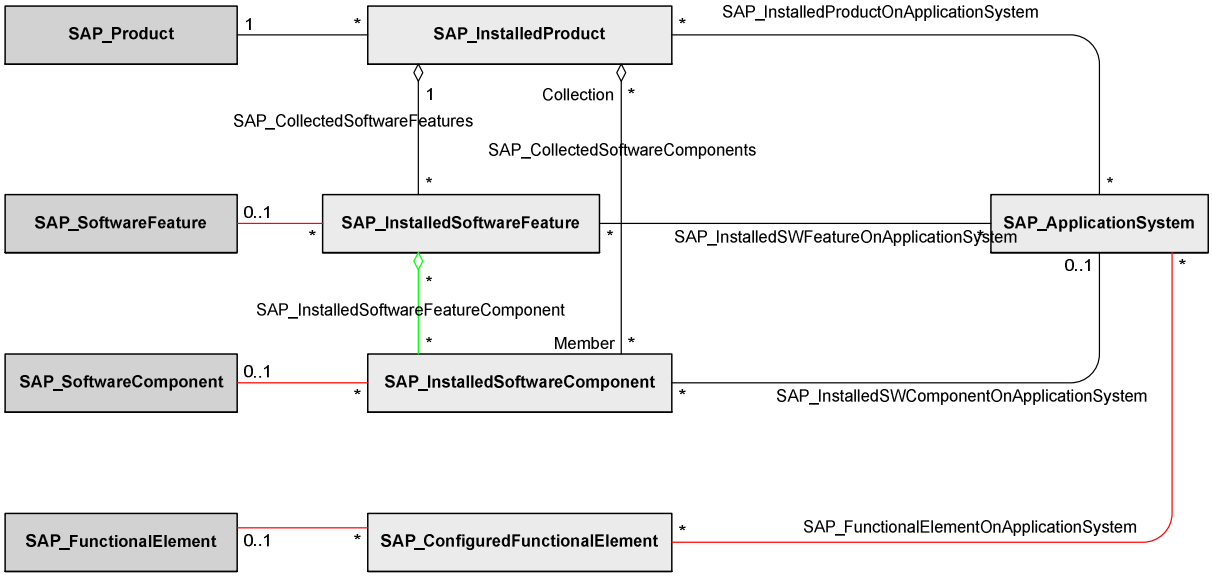

Figure 4: LM Meta Model Landscape Directory

Main Architecture Concepts and Decisions for Design-time

So far we focus on JEE as runtime. The standard development environment for this target runtime at SAP is Eclipse. We will implement a new Editor in Eclipse which allows the creation and change of artifacts of the LM Model.

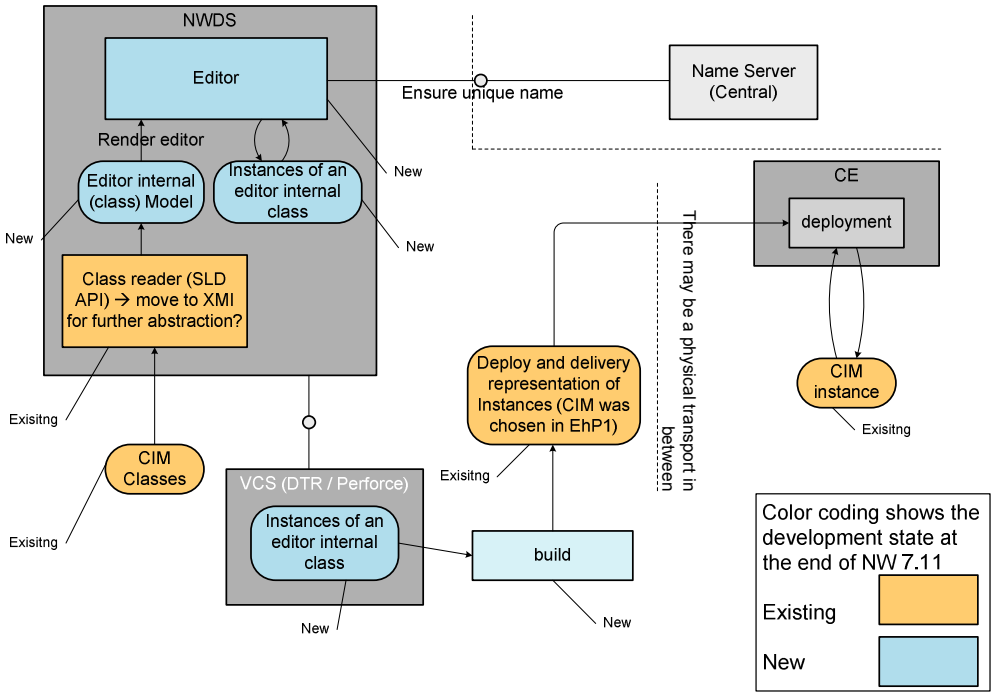

Figure 5: Design time architecture

For Eclipse based development at SAP the default structure organizing the different artifacts is described by the (Development) Component Model. Thus the entries carrying design time artifacts are development components (DCs) which are aggregated into software components (SCs). DCs in turn

are potentially compiled to deploy time artifacts called SDAs (Software Deployment Archive). SCs are potentially compiled to delivery time artifacts called SCAs (Software Component Archive – not Service Component Architecture as in the industry standard). The details of he Component Model and the SDA and SCA structure are explained elsewhere. EDEN introduces a new structure describing a yet un-modeled aspect of our software. The yet un-modeled aspect is the functionality that our software provides. The development and delivery process however will still be organized by the component model. Thus design time artifacts will be organized in DCs, compiled to SDAs as deploy time artifacts, aggregated to SCAs as delivery artifacts. The parts of the LM Model shown in Figure 3 will become design time artifacts. This will lead to the remarkable situation that a software component description is a design time artifact which is located in a development component[1] (as a general container of design time artifacts). The overall LM Model contains entries from all products produced at SAP and at ISVs or customers. The ownership obviously is distributed over the organization. The editor and the design time store of LM Model artifacts needs to support distributed development. This means working with incomplete data and associations pointing outside of a development component. End points of such associations are stored as a CIMReference so that the keys are known and the build is able to later fill in the real instance data from another DCs. See figures …

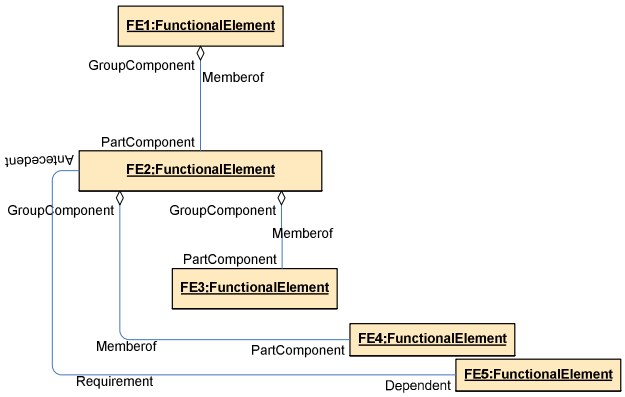

Figure 6: Example of Functional Elements related to each other through different types of associations

[1] This idea has been generalized by the Symbiosis project. They call development component containing data on a different software structure "structure components".

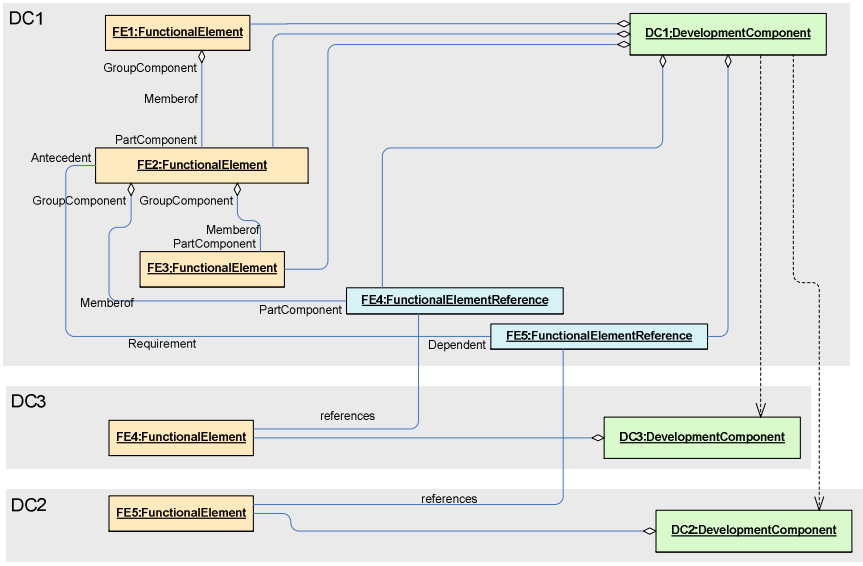

Figure 7: The example shown in Figure 6 with a mapping to the Development Components used to store the data at design time

In the example FE2 refers to FE4 with a “member of” association. FE4 however is defined in a different DC. The “member of” association is then stored in the DC holding the FE2 definition. However the association points to a proxy of FE4 which is shown as FE4 being an instance of FunctionalElementReference. The proxy is resolved with the real object at build time of DC1. If a proxy cannot be resolved a build error is created.

Main Architecture Concepts and Decisions (Runtime)

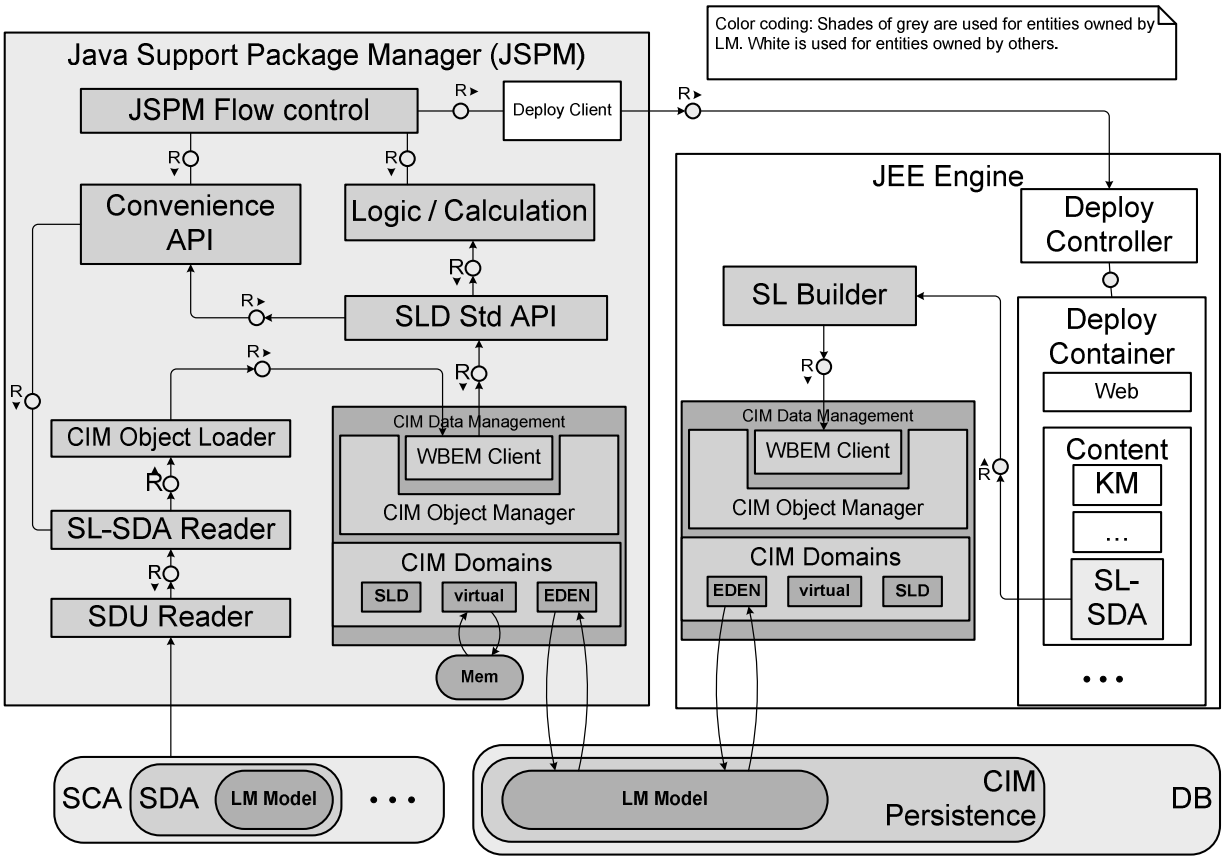

Figure 8: Architecture Overview Diagram (SL Runtime)

Figure 8 shows an architectural overview for the runtime of software logistics (SL) processes. The picture uses the Java Support Package Manager (JSPM) as an example of a SL tool operating on the LM Model. The central component handling the LM Model is the CIM Data Management. This component is part of the SL tools and the JEE engine. The task of the CIM Data Management is maintenance and persistence of LM Model according to the LM Meta Model. The main use case supported with the architecture diagram shown in Figure 8 is the update of the JEE system. The new software is represented in SCAs containing SDAs. One of these SDAs contains the new LM Model. JSPM reads this LM Model using the other blocks shown in the diagram and compares the Model with the data in the JEE system. Based on the comparison of the structure of the installed software and the structure of the new software it supports the user in deploying the right set of SCAs.

Component	*Description*	*Responsibility*
JSPM Flow Control	Controlling a JSPM process – user guidance, sequence of steps, calculations	LM SL SOF
SL-SDA Reader	Simplified read access for certain parts of an SDA.	LM SL
CIM Object Loader	Reads CIM XML and passes it to the WBEM Client	LM SL SLD
Convenience API	SL convenience API wrapping the SLD std API	LM SL SOF
Logic /Calculation	Calculation logic comparing the current software state of	LM SL SOF

	the JEE engine with the possible future state. Ensures completeness of the selected software being installed with respect to the LM Model.	
SLD Std API	Represents CIM classes (LM Meta Model) as Java Objects.	LM SL SLD
CIM Data Management	Maintenance and persistence of LM Model according to the LM Meta Model	LM SL SLD
Deploy Controller	Central channel for all deployment processes into the JEE engine. Distributes deploy task to Deploy Containers which are responsible for different software types.	JST Deploy
Deploy Container	Interface to be implemented by different deploy software responsible to handle different software types	JST Deploy
Web	An example of an implementation of the Deploy Container for Web Applications	JST Deploy
Content	An example of an implementation of the Deploy Container for content (having trivial engine life cycle).	JST Deploy
KM	A specialization of the Content Deploy Container for KM content.	KM
SL-SDA	A specialization of the Content Deploy Container for LM Meta Model and LM Model data.	

Total Cost of Ownership

The enhancements of the LM processes provided to the customers by EDEN are expected for lower cost of ownership by decreasing the costs of understanding and reducing the risks of errors when selecting a set of software for installation or update.

Total Cost of Development

By scaling the task of providing LM Model data over the whole organization we will most likely increase the costs of development a bit due to the fact that distributed development is always more expensive as compared to a central solution. However the development costs will also be able to scale out to the part of the organization causing the need for LM Model data (e.g. a new product being developed).

At the same time the editor capability should reduce the costs of developing certain parts of the LM Model by easing this process. Thus the total cost of development is expected to decrease.

Deployment

The vision is to have one EDEN store per SAP System technology (one for JEE, ABAP, Trex, LifeCache,…) . These stores provide the data for a central SLD. It is in discussion if an EDEN store

would be useful collecting all data of one system. Since an SAP system itself can have various different deployment layouts it will not be easy to provide a clear architecture for this system local EDEN store.

Maintenance

Standard maintenance procedures run by SL tools apply. The SL tools themselves have self update capabilities.

Architecture Documentation

- **ERD CE 7.11:** \\dwdf207\NWArchDef_archive\ERD\2007\2-Final\IB-02-Composition\EDEN ERD Format Review Corrected.doc
- **SRS CE 7.11**: https://ifp.wdf.sap.corp/sapspec?prefix=PR&id=0000000000000004994&version=work&sap-language=EN&sap-syscmd=nocookie
- **ERD NW 7.12** (in Review): \\dwdf207\NWArchDef\ERD\2008-Ehp2\2-InReview\Eden_Architectural_Overview\NW-ERD-EDEN-v1.0.doc