D42.2 –
Generic Service
Development Platform specifications
and architecture –
M21 issue
VERSION HISTORY

<table>
<thead>
<tr>
<th>VERSION</th>
<th>DATE</th>
<th>NOTES AND COMMENTS</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.1</td>
<td>18.6.2013</td>
<td>INITIAL DRAFT STRUCTURE AND TOC</td>
</tr>
<tr>
<td>0.4</td>
<td>26.6.2013</td>
<td>ADDITIONS TO DRAFT, RESTRUCTURING</td>
</tr>
<tr>
<td>0.7</td>
<td>4.7.2013</td>
<td>SILO REVIEW VERSION</td>
</tr>
<tr>
<td>0.8</td>
<td>5.7.2013</td>
<td>INTERNAL REVIEW VERSION</td>
</tr>
<tr>
<td>1.0</td>
<td>15.7.2013</td>
<td>SUBMISSION VERSION</td>
</tr>
</tbody>
</table>

DELIBERABLE PEER REVIEW SUMMARY

<table>
<thead>
<tr>
<th>ID</th>
<th>Comments</th>
<th>Addressed (✓)</th>
<th>Answered (A)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>The expected results of the &quot;code generation&quot; process are not explicited (entity model, CRUD web app / web services...), as well as the choices for the targeted programming language(s) (java?) and platform(s) (raw java beans, JPA persistence...?).</td>
<td>✓ (Additions to the relevant sections)</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Please, give an idea to what extent the usage of the GSDev platform will make the development work more productive / efficient than in a “traditional” development process / IDE. For instance, the deliverable does not mention: the amount of automatically generated code / the amount of manual work needed for the finalization of an application. Please, detail which UML diagrams types are exploited for automatic code generation.</td>
<td>✓ (Additions to the relevant sections)</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>There is no explicit mention of the &quot;TSM&quot; metamodel as it was defined in the D11.1 (&amp; D11.2). Will a specific &quot;TSM editor&quot; be developed and embeded within the GSDev platform?</td>
<td>A: Most “new” TSM level (i.e. not at the TIM level) IT information is covered by the data schemas and processes defined during the model enrichment process in the GSDev. For this reason, no explicit and separate TSM editor is planned. However, if necessary, the manipulation of model data inherited from the TIM level at the TSM level will be examined in coordination with SP4 and SP1 partners. ✓ Chapter 3 has been edited to focus on presenting the process of evaluation of programing environments relative to MSEE, leading to the selection of the Eclipse platform. The original content can be found in the Appendix.</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>No need to repeat section 3, related to SotA of programming environments (already presented in D42.1).</td>
<td>✓</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td><strong>5</strong></td>
<td>Please, justify the choice of WebDAV over other concurrent version systems for the model repository, which would leverage more advanced collaborative modelling / development scenarios.</td>
<td>✓ (Additions to the relevant sections)</td>
<td></td>
</tr>
<tr>
<td><strong>6.</strong></td>
<td>Additions to the Executive Summary for clarity</td>
<td>✓</td>
<td></td>
</tr>
</tbody>
</table>
TABLE OF CONTENTS

LIST OF FIGURES..................................................................................................................5
LIST OF TABLES.......................................................................................................................5
ACRONYMS .............................................................................................................................5
EXECUTIVE SUMMARY..........................................................................................................7

1 INTRODUCTION ....................................................................................................................8
  1.1 CONTEXT AND PURPOSE OF THE DELIVERABLE ......................................................8
  1.2 RELATION TO OTHER WORK PACKAGES AND DELIVERABLES .........................8
  1.3 STRUCTURE OF THE DOCUMENT ..............................................................................9

2 GENERIC SERVICE DEVELOPMENT PLATFORM REQUIREMENTS...............................11
  2.1 SYSTEM CONTEXT .........................................................................................................11
  2.2 PLATFORM OBJECTIVES ...............................................................................................12
  2.3 USE CASE ....................................................................................................................14
  2.3.1 Evolution of the main use case ................................................................................17
  2.4 REQUIREMENTS ..........................................................................................................18
  2.4.1 Functional Requirements .........................................................................................18
  2.4.2 Non-functional Requirements ................................................................................20
  2.4.3 Evolution from initial specification .........................................................................21

3 STATE-OF-THE-ART ANALYSIS OF CANDIDATE SERVICE DEVELOPMENT PLATFORMS
AND PLATFORM SELECTION ...............................................................................................23
  3.1 INTRODUCTION ..........................................................................................................23
  3.2 METHODOLOGY ..........................................................................................................23
  3.3 RESULTS .......................................................................................................................23

4 GENERIC SERVICE DEVELOPMENT PLATFORM ARCHITECTURE....................................25
  4.1 ARCHITECTURE ..........................................................................................................25
  4.1.1 Eclipse Runtime .......................................................................................................28
  4.1.2 Generic Development Platform Core ......................................................................29
  4.2 INTEGRATION WITH OTHER MSEE PLATFORM COMPONENTS ..................................44
  4.2.1 Integrated service development workflow ............................................................44
  4.2.2 Integration wizards ................................................................................................45
  4.2.3 Integration with the Mobile Business Platform ......................................................51
  4.3 FURTHER WORK TOWARDS THE FINAL PROTOTYPE (M30) ......................................51
  4.3.1 Model Repository ..................................................................................................51
  4.3.2 Generic Development Platform Core ......................................................................52
  4.3.3 Integration with other MSEE components .............................................................53
  4.3.4 General User Interface improvements ...................................................................54

5 CONCLUSIONS ...................................................................................................................55

REFERENCES ..........................................................................................................................57

APPENDIX ...............................................................................................................................59

STATE OF THE ART ANALYSIS OF VISUAL PROGRAMMING ENVIRONMENTS AND EVALUATION (D42,1 – M12) ........................................................................................................59
  JDeveloper .........................................................................................................................59
  Geany .....................................................................................................................................60
  IntelliJ IDEA ........................................................................................................................60
  Aptana .................................................................................................................................61
  NetBeans .............................................................................................................................61
  Eclipse .................................................................................................................................62

CONCLUSIONS ..........................................................................................................................63
LIST OF FIGURES

Figure 1: D42.2 Structure ........................................................................................................... 9
Figure 2: Generic Service Development Platform interconnections ........................................... 11
Figure 3: Service Development Platform Process Flow ............................................................... 17
Figure 4: Generic Service Development Platform Architecture ................................................. 26
Figure 5: Model Repository Architecture .................................................................................. 31
Figure 6: Model Repository Deployment .................................................................................... 32
Figure 7: Model Repository View plugin .................................................................................... 33
Figure 8: ATL-based model transformation, adapted from [10] .................................................... 35
Figure 9: Generic Model Transformation Workflow in MSEE’s Generic Service Development Platform ......................................................................................................................... 37
Figure 10: UML-to-code transformation workflow ................................................................. 39
Figure 11: UML-based TSM Model with Java bean stereotyping and data types ...................... 39
Figure 12: WSDL-to-code transformation workflow ............................................................... 40
Figure 13: BPMN-to-.BAR transformation workflow ............................................................. 41
Figure 14: UML Model Editing and enrichment in the Development Platform environment ....... 42
Figure 15: BPMN Model Editing in the Development Platform environment ........................... 43
Figure 16: Java Source Code generated by the Transformations Plugin for the Indesit use case ... 44
Figure 17: Fully integrated Design-Development-Service Registration process within MSEE ...... 45
Figure 18: Import/export functionalities of the Generic Service Development Platform .......... 45
Figure 19: BSM, TIM and TSM modelling levels ...................................................................... 46
Figure 20: Running the TIM Models Import Wizard ............................................................... 47
Figure 21: Connecting to the Model Repository ........................................................................ 48
Figure 22: Selecting TIM models to import ............................................................................. 48
Figure 23: Selecting workspace location to import models ...................................................... 49
Figure 24: Searching for service categories ............................................................................. 50
Figure 25: Service Discovery results ....................................................................................... 51

LIST OF TABLES

Table 1: Relation of D42.2 with other MSEE Deliverables .......................................................... 9
Table 2: Mapping of MDA elements into MSEE ..................................................................... 13
Table 3: IDE Comparison Overview (I) .................................................................................... 24
Table 4: IDE Comparison overview (II) .................................................................................... 24
Table 5: Model Repository Client API ...................................................................................... 33
Table 6: Development components and the transformation process ....................................... 38

ACRONYMS

API: Application Programming Interface
IDE: Integrated Development Environment
IEP: Innovation Ecosystem Platform
IPMS: Ideation Process Management System
M2M: Model-to-model transformation
MMT: Model-to-model transformation
M2T: Model-to-text transformation
PDE: Plug-in Development Environment
SLM: Service Lifecycle Management
SPx: MSEE Sub Project x
TE: Transformation Engine
TIM: Technology Independent Modelling
TSM: Technology Specific Modelling
UI: User Interface
USDL: Unified Service Description Language
VCS: Version Control System
WPx: MSEE Work Package x
Executive Summary

The MSEE Generic Service Development Platform is an integrated development environment, which enables IT users to develop software applications and services for MSEE. This platform will be used by developers inside a Manufacturing Service Ecosystem to develop applications and services starting from existing components (e.g. starting from SP3 components and services to develop SP6 test case applications). In the particular case of servitisation, the platform is used to implement the final transformation steps concerned with the design and implementation of an IT service or application in a Virtual Manufacturing Enterprise (e.g. starting from SP1 VNME models to develop the IT section of a servitisation), i.e. the transformation of Technology Independent Models of services and applications into Technology-Specific Models (targeted to specific technologies and platforms), and, finally the generation of executable code. This process is based on the Model-Driven Service Engineering approach defined in SP1 and based on the principles of the OMG Model Driven Architecture.

The present document, D42.2, is the second deliverable of Task T42.1 “Design of Generic Service Development Platform”. Task T42.1 aims to describe and detail the specifications and architecture of the platform, in its first and final implementations. The first deliverable of T42.1, D42.1 provided an initial specification for the first prototype delivered in M18 as D42.3. Deliverable D42.2, on the other hand, aims to update this specification, based on the experience gained from developing the first prototype, further research, and cooperation with stakeholders involved in the creation of associated MSEE platform components. This specification will focus on the final release of the platform on M30 (D42.4 “Generic Service Development Platform final prototype”)

As described in this document, beyond its generic purpose the main use case for the platform is in the servitisation cycles and involves the importation of Technology Independent Models (created by methods and tools developed in Work Package 15), as well as the selection of the desired target technologies and components. Through a “model enrichment” process, the Technology-Independent Models are edited and annotated with metadata derived from the desired target technology. This process produces a Technology-Specific Model, which can be then used to generate executable service code. These services are registered into the Service Delivery Platform, making them available for use in the appropriate MSEE-supported ecosystems through the Innovation Ecosystem Platform of SP2.

This way, the Generic Service Development Platform forms the final link in the application and service development chain, by using the products of the modelling processes in SP1 and SP2 to build the final executable applications, services and service components.

Based on the requirements set by the anticipated use of the platform, a range of potential candidate development environments for the necessary technology foundation have been taken into consideration and were thoroughly examined. To this end, the platforms that have been examined are the ones that are in a position to support features required by the MSEE pilots such as visual programming interfaces, support for modelling, collaborative development, documentation and support.

Through this investigation, the architecture for the Generic Service Development platform has been based on the Eclipse development environment that has been selected as the ideal match for these requirements. By using existing plugins and developing new Eclipse-compatible components, the Generic Service Development Platform is able to support graphical modelling, model transformation and collaborative development while enforcing user access privileges and data protection, as well as integrating with the rest of the MSEE platform environment.
1 Introduction

1.1 Context and Purpose of the Deliverable

According to MSEE Project – Description of Work (2011), work package WP42 aims to create “a visual programming environment that will facilitate the development of complex front-end applications, involving execution of relatively complex business processes that rely on back-end services business users”. This environment is MSEE’s “Generic Service Development Platform”.

Deliverable D42.2 contains the final results of Task T42.1, “Design of Generic Service Development Platform”. It is an update of the initial platform specification document (D42.1 “Generic Service Development Platform specifications and architecture – M12 issue”) and is based on the experience from the development of first prototype (D42.3 “Generic Service Development Platform first prototype – M18”), as well as collaboration with MSEE stakeholders and the development of other parts of the MSEE platform.

The Generic Service Development platform will include models and tools necessary for a user to model, simulate, test and execute business innovation artefacts. These artefacts can be product/service specifications, skills and competencies, decisional and business processes, legacy data and enterprise systems.

The specifications in this document will be used for the development of the final version of the platform (task T42.2 “Development of Generic Service Development Platform”) and to anticipate the work required in Task T42.3 “Integration and Testing”.

Task T42.2 has been organised in two cycles, the first being focused on the development of D42.3 “Generic Service Development Platform first prototype - M18” and the second to the development of D42.4 “Generic Service Development Platform final prototype - M30”.

1.2 Relation to other Work Packages and Deliverables

The Generic Service Development platform will interoperate directly with platforms and tools developed in WP15 (the SLMtoolbox)[2], WP43 (the Service Delivery Platform) and WP44 (the Mobile Business Platform) and indirectly with those developed in WP26 (the Innovation Ecosystem Platform).

Also, this document is based on the results of WP41 “MSEE Service-System Functional and Modular Architecture”. The contents of this deliverable will be used by deliverable D42.4 “Generic Service Development Platform final prototype - M30”.

In this context, the relations of the deliverable in hand with other deliverables of the project are shown in the following table:

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Relation to D42.2</th>
</tr>
</thead>
<tbody>
<tr>
<td>D41.1: “MSEE Service-System Functional and Modular Architecture”</td>
<td>Initial functional and modular architecture about MSEE’s innovative platforms, including refined analysis usage scenarios and public APIs.</td>
</tr>
<tr>
<td>D15.3: “Specifications and Design of SLM Platform”</td>
<td>D42.2 is referencing D15.3 (the updated version of deliverable D15.2) while examining particular use cases and analysing their elements and requirements</td>
</tr>
<tr>
<td>D42.1: Generic Service Development</td>
<td>The initial version of this document</td>
</tr>
</tbody>
</table>
Platform specifications and architecture

<table>
<thead>
<tr>
<th>Deliverable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>D42.3</td>
<td>Prototype based on the specification included in D42.1.</td>
</tr>
<tr>
<td>D42.4</td>
<td>Final Prototype based on the updated specification in D42.2.</td>
</tr>
<tr>
<td>D43.2</td>
<td>Updated version of the MSEE Service Delivery platform architecture. Products of the development process are registered with the Service Delivery Platform.</td>
</tr>
<tr>
<td>D44.2</td>
<td>Updated version of the MSEE Mobile Business platform architecture. The Generic Service Development platform and the Mobile Development module of the Mobile Business platform interoperate in development activities targeting mobile devices.</td>
</tr>
<tr>
<td>D26.2</td>
<td>Updated version of the MSEE Innovation ecosystem platform architecture. The Generic Service Development platform produces Service Deployment Artefacts which can be used by the Innovation Ecosystem Platform.</td>
</tr>
</tbody>
</table>

Table 1: Relation of D42.2 with other MSEE Deliverables

1.3 Structure of the Document

![D42.2 Structure](image)

**Figure 1: D42.2 Structure**

**Block A** aims to introduce the reader to the purposes of this deliverable, and examine the objectives and the main requirements of the Generic Service Development platform. Chapter 1 “Introduction” contains an introduction to the context, objectives, associations and structure of this document. Chapter 2 “Generic Service Delivery Platform Requirements” contains an extensive analysis of the requirements that govern the development of the Generic Service Development platform.
Block B focuses on the technology foundation of the Generic Service Development platform. Chapter 3 “State-of-the-art analysis of candidate Service Development Platforms and platform selection” presents an account of the process used for the selection of appropriate technologies for the implementation of Generic Service Development platform in relevance to MSEE’s purposes and needs.

Block C synthesises the results from the previous chapters into a proposed architecture for the Generic Service Development platform, while discussing notable conclusions from this work. Chapter 4 “Generic Service Development Platform Architecture” presents an analysis of the proposed architecture of the platform, including detailed descriptions of the proposed components and its interdependencies with other components of the MSEE platform. Finally, Chapter 5 “Conclusions” concludes the deliverable and discusses the necessary next steps.
2 Generic Service Development Platform Requirements

In this chapter, we present an overview of the requirements for MSEE’s Generic Service Development Platform.

2.1 System context

The Generic Service Development Platform is a fundamental component part of the MSEE platform. Along with the SLM Toolbox[2], provides the main method for MSEE users for the:

“...the development of complex front-end applications, involving execution of relatively complex business processes that rely on back-end services business users. The Development Platform will allow the design of the different interfaces which users will interact with so that their users will be able to define the relationships between them and with the resources and services provided by the back-end.”[1]

The Generic Service Development Platform will be interconnected with the rest of the MSEE platform as depicted in the following figure:

![Diagram of Generic Service Development Platform interconnections]

Figure 2: Generic Service Development Platform interconnections

In brief, the SLM Toolbox is used to create high-level TIM models of services and upload them to a common Model Repository implemented as a part of the Generic Service Development Platform. These models are then imported by the Generic Service Development Platform. Using these models, developers instantiate executable services and register them with the Service Delivery Platform. In reference to the TSM metamodel defined in D11.2[20], the Generic Service Development platform manipulates the IT parts of the model, and specifically, defines and adds Enterprise Application functions and the necessary Information Schemas.

Also, during development, users of the Generic Service Development Platform will be able to search for and reference services already registered with the Service Delivery Platform. Finally, the Development platform will support the Mobile Business Platform in the process of Mobile Development, by providing UML models and generate code for the mobile domain.

A further indirect connection exists with the Innovation Ecosystem Platform, without direct interoperability between the two platforms. As part of its normal flow of operations, the Development platform produces Service Deployment Artefacts such as executable Business Archives that can be uploaded to and executed by the Innovation Ecosystem Platform.
These interconnections represent the core added value of the Generic Service Development platform. The Generic Service Development platform provides a custom environment for a service development workflow in concert with MSEE platform actors, processes and resources. This achieved through integration with other MSEE components, as well as a customised IDE providing tools for MSEE service development in a unified environment.

### 2.2 Platform Objectives

The different components of the MSEE platform are designed with a view to support the concept of the service system life cycle through its various stages.

As analysed in [1], the Generic Service Development platform, combined with the MSEE Service Lifecycle Management Toolbox[2] will support the following stages of the Service System Life Cycle:

1. **Service System Requirements**: identify, describe and model end-users requirements for the service system
2. **Service System Design**: design, specify and simulate the system that will provide that service
3. **Service System Implementation**: describe how the designed service system will be realized, deliver and implement physically all the needed components,
4. **Service System Operation**: service system is operational for use by customers and by all the stakeholders. This includes service consumption and interaction with customers, monitoring, evaluation and maintenance.

Stages 1 and 2 are directly supported by the SLM toolbox, which interoperates closely with the development platform. Stage 4 is indirectly supported, since the SLM Toolbox is used to define aspects of the governance of the service (such as monitoring and control). The SLMToolBox implements the first phases of service engineering, defined in the servitization process steps described in D11.1 – “Service concepts, models and method: Model Driven Service Engineering”[20].

The SLM Toolbox, although an integral part of the Service Development process and in close cooperation with the Generic Service Development platform, is described in further detail in deliverable D15.3 “Specifications and Design of Service Lifecycle Management Toolbox” [2].

As stated in D15.3, The SLM Toolbox is

“An integrated modeling tool, dedicated to manufacturing services lifecycle management. It will allow:

- To take benefit of a model based architecture: syntactic validation; transformation; execution; …
- Maintain the coherence through the whole engineering process - from Business requirements to IT implementation (modeling)
- Anticipate / simulate the result of the service (engineering)
- Design the governance of the service (monitoring & control)”

However, the SLM toolbox will not provide support for implementation/coding of software components, the creation of Business Intelligence reports and the execution of service governance mechanisms.[2]
This leaves stage 3, “Service System Implementation” as the main function of the Generic Service Development platform that will be implemented in WP42 and analysed in depth this deliverable.

In MSEE, the stages of Requirements, Design and Implementation will be based on model enrichment through extensions combined with transformation and mapping mechanisms, mainly implemented through the use of the Model-Driven Service Engineering Architecture, defined in Deliverable D11.1[20] which, in turn, is based on concepts from MDA (Model Driven Architecture)[21]. Using an MDA-derived approach, the Generic Service Development platform will allow users to easily adapt service and application designs to target platforms/ technologies, allow model-level development and design and provide automation (in the form of model transformations and code generation).

For the Implementation stage, the MSEE Generic Service Development Platform architecture aims to closely follow the MDA approach for software design which is comprised of the following stages [6]:

- Specification of a PIM - Platform Independent Model.
- Specification of Platform Models (PM) for specific target platforms
- Selection of a specific Platform Model for the system.
- Transformation of a PIM into a PSM, using the selected Platform Model
- Transformation of the PSM into software system code.

Within MSEE there is a distinctive mapping between the standard MDA models naming and the MSEE-specific models, defined in [6] and Deliverable D11.2 [20] and summarised below:

<table>
<thead>
<tr>
<th>MDA Acronyms</th>
<th>MSEE Naming</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computation Independent Model (CIM)</td>
<td>Business Service Model (BSM)</td>
</tr>
<tr>
<td>Platform Independent Model (PIM)</td>
<td>Technology Independent Model (TIM)</td>
</tr>
<tr>
<td>Platform Specific Model (PSM)</td>
<td>Technology Specific Model (TSM)</td>
</tr>
<tr>
<td>Platform Model</td>
<td>Technology Model</td>
</tr>
</tbody>
</table>

Table 2: Mapping of MDA elements into MSEE

In the case of MSEE, the “Service System Implementation” stage corresponds to the transformation of a “Technology Independent Model” of a service to a “Technology Specific Model”. TSM models need to be instantiated with executable services and applications.[20] These may originate from the Generic Service Development Platform itself, the Service Delivery Platform or the Open Internet layer of the MSEE platform, and they may correspond to existing services/applications or be entirely new software.

In the Generic Service Development Platform the generic process flow begins with an initial high-level composition of the required process elements, corresponding to a Technology Independent Model. The TIM is mapped to a TSM model using input from a “Technology Model”, i.e. information governing the mapping to specific technologies and implementation. The TSM could also be, to an extent, executable, and allow test-runs and debugging. The TSM is then processed via a development environment, which results eventually in an executable build of the required implementation, ready to be deployed to the run-time environment targeted by the System Model selected for the TIM to TSM transformation.

This process is consistent with the goals of the Generic Service Development, and the MSEE ecosystem service development process in general. For this reason, we shall use this concept for the functionality of the Generic Service Development Platform, in order to begin to explore the basic requirements for its operation and implementation.
2.3 Use Case

The main use case for the Generic Service Development platform is the process of transforming a TIM received from the SLM Toolbox (via a Model Repository) into a TSM and ends with the generation of executable code and, finally, registration of the model with the Service Delivery Platform. This process flow is a refinement and evolution of the flow described in D4.1.1 [1] and D42.1 [22].

The use case is described in detail below

Actors

The main use case actors are expected to be technically qualified service analysts and service developers.

Inputs

A TIM model: The TIM model may be generated by the SLM Toolbox or be imported via other means. The TIM model is imported through the Model Repository

Various elements from the Model Repository, Service Development Platform-registered services, other external software development components

Outcome

A new service is registered with the Service Delivery Platform

Environment

Interface: A graphical user interface allowing point-and-click operations.
Model repository: a repository containing various assets used in the development process, such as intermediate artefacts, components, etc.
Transformation Plugin: a software component that performs mapping from TIM to TSM Platform and mapping from service compositions to source code.

Main Flow

The main flow illustrates the central features of the core Generic Service Development Platform use case:

1. The user initialises the “Import TIM model from repository” process of the Generic Service Development Environment
2. The user selects a Technology Independent Model from the Model Repository for importation to the Generic Service Development Platform
3. The user selects a technology platform (a Technology Model) that will determine platform specific parameters for the transformation process.
4. At any point, the user may search for and choose to reference various external components:
   - Components stored in the Model Repository of the Generic Service Development Platform
   - Services registered with the Generic Service Development Platform
   - External elements
5. The user engages in the “service authoring” process: using a graphical point-and-click
interface and text input, the user edits the TIM model.

- The user can fully edit the imported TIM model, adding and removing elements, changing element interconnections and properties, referencing external components etc.
- The user can import fragments for this specific platform from the Model Repository and Service Delivery Platform
- Users are not required to build a single, complete, end-to-end implementation of a service. They may choose to instantiate, e.g. a collection of elements or service fragments for reuse

6. The user engages in the “model enrichment” process: using a graphical point-and-click environment, or automated processes, the user “enriches” the technology-agnostic TIM model with annotations derived from the selected Technology Model, thus producing a TSM model.
   - Depending on the source modelling language and the target technology, some parts of this process may be automated – for example, if the model and target language use abstractions that can be mapped to each other.
   - Based on the selected Technology Platform, the development environment may provide technology-specific functions and interface elements.

7. After completing the authoring process, the user requests the generation of the TSM model source code.

8. The Generic Service Development platform uses the TSM model to produce technology-specific code and other artefacts.
   - Some components may, in the end, be represented as, e.g. empty classes. For example, some components may only be associate with technology-specific code templates, or may have been specified for the first time during the service authoring process. In this case, the application produces an editable source code framework or template based on the TSM.

9. The user edits, tests and debugs the source code or source code framework.
   - From a technical standpoint, this step is similar to software development activities as performed in IDEs in general, and will not be further analysed at this stage. They involve editing, building executables, test runs, debugging, etc.

10. The user compiles the code into an executable build and deploys the service. (the deployment process is beyond the scope of this deliverable and is not further analysed)

11. The user requests registration of the newly-built service with the Service Delivery Platform

12. The service is registered with the Service Delivery Platform and, thus, becomes available to the system.

**Extensions**

Extensions are possible deviations from the main flow.

At steps 1-2
- The development process can start without the importation of a TIM model. At step 5 the user creates a new service model from scratch. The development process can proceed as described. Steps 1,2 are omitted.

At steps 1-5
- The development process starts with the importation of specific external components, to be customised for use within MSEE. Steps 6,7,8 may be omitted if the external components are source code.

At step 9
• The result of automatic generation may not require further editing. All necessary input has been provided by the previous development stages. Step 9 is omitted.

At steps 11-12
• The generated product does not necessarily have to be registered with the Service Delivery Platform. In this case, it is a stand-alone application, component or service description / orchestration. Steps 11, 12 are omitted.

At any point:
• The user may save the entire service project to a personal workspace or in a common workspace in the case of collaborative development. The project includes
  o The TIM model
  o Technology specific selections
  o References to external components
  o Generated TSM Platform
  o Service composition
  o Generated source code
  o Appropriate metadata (such as Service Delivery Platform registrations, project owners, project editors)

• The user may choose to register various intermediate products and artefacts to the Model Repository. Registration will make these available to other users, although access may be subject to specific privileges. These artefacts include:
  o Edited TIM models
  o TSM models
  o TSM source code
  o Other resources as necessary

The main flow is represented in the following diagram:
2.3.1 Evolution of the main use case

The main use case has evolved since the original specification in D42.1[22]. The main differences lie in the execution of the core functionality of the Generic Service Development Platform, that is, the transformation from TIM to TSM models.

First, it was decided to eliminate the “TSM-platform” concept [22]. The “TSM-Platform” was initially specified as an intermediate product of the transformation process. The user would import a TIM model and select the appropriate target Technology Platform. Afterwards, the platform would generate a “TSM-Platform” through an automatic process. The TSM-Platform would still be a model of the process, (i.e. not source code) but it would incorporate elements derived from the target Technology Platform. For example, it would include references to code libraries, functions to apply technology specific constructs, etc.

However, explicitly requiring an intermediate TSM-platform product has proven to be problematic. In essence, the TSM-platform serves no purpose other than setting up a service development template according to information about the technology model and the TIM model. Therefore, apart from any customisations based on the particular application, this can be achieved by simply selecting the desired Technology Platform and aided by some degree of automation in preparing the user workspace.
Also, the model transformation activities have been separated into two elements instead of a single “service authoring” step. Now, in the “service authoring” step, the user simply edits an imported TIM-level model. In the “model enrichment” stage, the user annotates the elements of the TIM model and creates the TSM model, either manually or through some degree of automation depending on the model input and the target technology. This change aligns the use case with the research results from the prototype development while improving clarity, thus facilitating the requirements analysis process.

2.4 Requirements

The basic usage of the Generic Service Delivery Platform, as presented in the previous section, leads to a set of functional and non-functional requirements.

This set of requirements will follow IETF RFC 2119 guidelines “Key words for use in RFCs to indicate Requirement Levels. For this deliverable, the keywords “Recommended” and “Optional” have been selected while “Required” is implied unless otherwise specified by the other two keywords.

2.4.1 Functional Requirements

User Interaction

- The application will provide a graphical user interface
- All functions related to the operation of the application will be accessible via this single unified interface.
- The user interface will present a unified view of the development process, providing access to all artefacts included in a single development project
- Service Composition will take place in a point-and-click interface,

Basic Operations

Main use case

- The application will support importing TIM models
  - From the Model Repository
  - Recommended: As a separate file/group of files
  - The application will provide support (at the minimum) for UML, BPMN and WSDL (SOAP web services)
  - Recommended: Support for RESTful services
- The application will support the selection of a Technology Model and setting appropriate parameters for the targeted platform
  - Recommended: Target technology platforms will be Java (for source code), JAX-WS (for web service development), Activiti BPM (executable Business Archive files for exports to the Innovation Ecosystem Platform and process execution)
  - Optional: RESTful services, other platforms as required
- The application will support searching for additional resources to include in the TSM
  - From the Model Repository
  - From the Service Delivery Platform
  - Optional: Other platform sources

1 IETF RFC 2193, Key words for use in RFCs to indicate Requirement Levels, Available at http://www.ietf.org/rfc/rfc2119.txt
- The application will import these elements into the service development project and will handle referencing.
- The application will provide a way for the user to edit the service and its components at the TIM level.
- The applications will provide a way for the user to annotate the TIM model using elements derived from the selected Technology Model, thus producing a TSM model.
  - **Recommended:** Provide support for the TIM modelling languages recommended in D11.2 [20]: UML, BPMN, WSDL.
- The application will generate the TSM source code based on:
  - The structure of the TSM model
  - The technology-specific annotations on the TSM model
  - The selected Technology Model and related parameters set by the user
  - Other options and parameters set by the user.
- The application will provide the means to edit, test, debug the source code and build executables.
- The application will allow the registration of a completed service, service component or collection of components with the Service Delivery Platform.
- **Recommended:** The application will provide automated features to assist the service development process. These may include:
  - Automated TIM annotations based on the selected Technology Model
  - Auto-complete features
  - Easy access to re-usable or common resources
  - Other capabilities.

**Extensions**

- The application will allow saving the developer’s work in various stages of completion along the course of the main use case in the form of “projects”.
- The application will allow registering the products of the development process (including intermediate products, e.g. service compositions) in the Model Repository.
- The application will allow the reiteration of the development process steps, e.g. the user returning to refine a service composition after inspecting the generated source code, etc.
- The application will allow performing the development process without an initial TIM model i.e. the development of services and applications from scratch.

**Integration and Interoperability with other MSEE Elements**

The application will need to communicate with the following external components:

- The SLM Toolbox, indirectly via the common Model Repository.
  - The application will support the importation of TIM models by parsing the SLM Toolbox output.
- The Service Delivery Platform, via the SDP API.
  - The application will be able to register services with the Service Delivery Platform.
  - The application will be able to search for services and service components in the Service Delivery Platform, in order to reference them during the development process.
- The Mobile Business Platform.
  - Generation of XMI representations of business objects for use with the Mobile Development Module.
  - **Recommended:** The application will be able to provide access to the Mobile Development module developed in WP44.
Collaboration and Workspaces
The Generic Service Development platform will provide tools for collaborative development. The Collaboration functionalities will be provided through the shared workspace of the Model Repository. The Model Repository will include:

- A Common repository and workspace for managing the development process, from the business to the technical level
- Locking and data integrity mechanisms to facilitate work on common resources
- User access control with user privileges depending on role, team, organisation, ecosystem (minimum)
  - Recommended: Access control will be based on the Federated Single Sign-On Utility service developed in WP33[26], to ensure consistency across the MSEE platform.
- Search capabilities on common resources, search pool depending on user privileges
  - Recommended: User privileges will be derived from MSEE’s Federated Single Sign-On Utility Service
- Recommended: Version control
- Recommended: Activity Logging
- Optional: Other capabilities as necessary

2.4.2 Non-functional Requirements

Usability
IDE design is now fairly mature, and provides several useful UI conventions, such as tooltips, syntax highlighting, workspace management, user interface organisation, etc. Following these conventions allows users to experience a shorter learning curve when encountering the development platform for the same time, while increasing productivity and ease of use.

Integration of different technologies and standards
From the processes and requirements defined above, it is obvious that the Generic Service Development Platform will integrate artefacts and processes based on various standards and technologies, e.g. graphical service compositions, source code in various programming languages, metadata, technologies etc. Also, manufacturing service applications may operate over a variety of platforms and frameworks (e.g. desktop machines, servers, mobile devices, IoT devices, web services, etc). The IDE and the Generic Service Development Platform should be able to support integrated development of complex applications based on heterogeneous technologies and standards.

Flexibility
As noted above, the Generic Service Development platform should be able to provide varied functionalities and may have to serve various technology platforms, implementation styles and standards. The IDE should be able to be upgraded and adapted to business needs without incurring significant overheads, e.g. for the inclusion of a new target platform or a programming language. For this reason, a modular technology base and platform are recommended.

Interoperability
The Generic Service Development platform will need to interoperate with other components of the MSEE platform, mainly the Service Delivery Platform, the SLM Toolbox, and the Mobile Business Platform. The development platform will need to be designed and implemented in order to ensure complete interoperability with these components, and should adhere strictly to API specifications and data formats.
Robustness and Error handling
As noted in the requirements, the development receives input from several sources, as well as providing a development, testing and debugging environment. The Generic Service Development platform should be able to handle malformed/unexpected inputs, interface problems, and various execution and runtime errors gracefully.

- Input checking for adherence to standards
- Descriptive error messages
- Graceful/ graduated failure states
- User workspaces and artefacts should be backed up automatically during normal operations
- Recommended: Failure states should provide means to return to normal operation

Security and access control
The Generic Service Development platform is a powerful tool within the MSEE platform, allowing the editing and registration of new ecosystem services and applications with the Service Delivery Platform. Also, the Model Repository and the SDP may contain several proprietary and confidential assets, such as source code, service descriptions, etc. Finally, the platform is expected to provide functionalities including service development and access to resources to several users, from different organisations/ ecosystems and with varied levels of responsibility.

For this reason, the platform is required to provide ways to distinguish access levels to functions and resources, e.g. the ability to register services to the Service Delivery Platform or access a certain group of model repository resources should be subject to authorisation. Furthermore, it is recommended that the platform should leverage the capabilities of the MSEE Single Sign-On Utility Service (developed in WP33) in order to determine user privileges and access to resources.

2.4.3 Evolution from initial specification
Along with the main use case, the platform requirements have been updated and refined based on further design work and the development of the first prototype (D42.3).

Starting from the user interaction side, it was decided to remove the requirement that the service authoring interface and process should be modelled after the Service Component Architecture (SCA) paradigm. The main reason for this change was that it became clear that input would be available to the platform in the UML, BPMN and WSDL formats. TIM models are primarily described using UML and BPMN, while services registered with the Service Delivery Platform are available as WSDL (in the first prototype). The preliminary investigation of the BSM-TIM-TSM transformation process in SP1, (as recorded in deliverable D11.2[20] – M12 ) recommended the use of these three standards for TIM-level models, while substituting BPEL for BPMN at the TSM level. As a result, subsequent work in SP4 focused on these modelling languages. Additionally, mature and open-source software components exist for these languages that can handle service model editing as well as integration with the Generic Service Development Platform.

Insisting on using SCA would automatically require the development of a transformation engine that would transform UML, BPMN and WSDL input into SCA models, as well as customising the relevant Eclipse plugin.

http://www.oasis-open.org/SCA
Therefore, it was decided to drop the requirement for strict SCA compliance, as the objective of service composition can be achieved with simpler and more flexible tools, while saving design and development effort for more critical issues (such as integration with other actors and the model transformation process). At the same time, explicit requirements for UML, BPMN and WSDL support have been added, as well as a recommendation for support of REST-ful services, as described in deliverable D43.2 “Service Delivery Infrastructure Specifications and Architecture”[3]. Further details on the evaluation of SCA and the implementation of the service authoring environment can be found in section 4.1.2.2.1.2 of this document.

Also, the requirements were adapted following the removal of the TSM-platform concept and the restructuring of the service development phase into two stages (service authoring, model enrichment). A further addition was the inclusion of a recommendation for various automated features in order to assist the development process. The “flexibility” requirement was also expanded, in order to make clear that it essentially calls for a modular platform.

Finally, a requirement was added for including access control functions using the Federated Single Sign-On Utility Service, developed in WP33[26]. This Utility Service aims to provide single sign-on functionality across the MSEE platform. The service allows for the implementation of a platform-wide access control scheme, ensuring a consistent security policy for all users and integrated components. For this reason, the required Model Repository, as well as the Generic Service Development Platform in general should take advantage of this functionality.
3 State-of-the-art Analysis of Candidate Service Development Platforms and platform Selection

3.1 Introduction

During the initial design stages (leading up to the initial specification in D42.1) various Programming Environments have been examined with the ultimate goal of selecting a technology foundation for MSEE’s Generic Service Development Platform.

The analysis was focused on the functional characteristics of each platform and attempted an evaluation of its relevance to MSEE and the Generic Service Development platform specifically and was presented in full in deliverable D42.1.

This chapter provides a brief account of the data collection and evaluation process, leading to the reasoning behind the selection of the Eclipse platform as a technology base for the Generic Service Development platform. The full analysis, as included in deliverable D42.1, can be found in the Appendix.

3.2 Methodology

The programming environments examined in the initial analysis were:
- JDeveloper
- Geany
- IntelliJ IDEA
- Netbeans
- Eclipse

These environments had been selected as potential candidates for the technology foundation, as they are general development graphical IDEs, enjoy a fair amount of support and are freely available.

For each environment, the main functional characteristics have been examined and a brief evaluation of its potential relevance to the Generic Service Development platform was performed. This evaluation was based on the following criteria:
- Functionalities (native or as plugins) relevant to the purposes of the Generic Service Development Platform
- The presence of collaborative development features
- Support and documentation, either from the originating organisation/developers or through a user community

3.3 Results

A two-stage comparison has been performed with the intent of determining the optimal fit. Initially, the existing technologies were compared with respect to their License, OS/platform support and features like debuggers, Graphical User Interface builders and version control features. This comparison is summarised in the table below:

<table>
<thead>
<tr>
<th>IDE</th>
<th>Developer</th>
<th>Latest Stable Release</th>
<th>Platform</th>
<th>License</th>
<th>Debugger</th>
<th>GUI Builder</th>
<th>VCSSupport</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDeveloper</td>
<td>Oracle</td>
<td>September 2012, 11.1.2.3.0 Build 6276.1</td>
<td>Multiple</td>
<td>Proprietary OTN JDeveloper License (Freeware)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, EP3</td>
</tr>
<tr>
<td>Geany</td>
<td>Colomban</td>
<td>June 2012,</td>
<td>Multiple</td>
<td>GPL</td>
<td>Yes, EP</td>
<td>No</td>
<td>Yes, EP</td>
</tr>
</tbody>
</table>

3 EP: Extra Plugin required
The available environments were then compared with respect to documentation and community support, functions/plugins relevant to the Generic Service Development Platform and their collaborative characteristics.

<table>
<thead>
<tr>
<th>IDE</th>
<th>Documentation and community support</th>
<th>Functionalities relevant to Development Platform</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDeveloper</td>
<td>Extensive</td>
<td>Limited</td>
<td>Oracle Team Productivity Software</td>
</tr>
<tr>
<td>Geany</td>
<td>Small</td>
<td>Very limited</td>
<td>Not supported</td>
</tr>
<tr>
<td>IntelliJ IDEA</td>
<td>Moderate</td>
<td>Limited</td>
<td>JetBrains-developed Team City, Youtrack</td>
</tr>
<tr>
<td>Aptana</td>
<td>Moderate</td>
<td>No</td>
<td>Git integration only</td>
</tr>
<tr>
<td>NetBeans</td>
<td>Moderate</td>
<td>No</td>
<td>Native Support</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Extensive</td>
<td>Yes, MDA, Automated Code Generation, Models and tools to model, simulate, test and execute business artefacts</td>
<td>Several plugins, providing versioning control, workspaces, etc</td>
</tr>
</tbody>
</table>

From the analysis performed above, the Eclipse IDE appeared to be the best candidate for the technology foundation of the Generic Service Development platform. It combines the following set of characteristics:

- Open source.
- Visual Programming Environment
- Open plugin architecture allows extensive customisations with minimal integration overheads.
- Modularity, including an extensive range of plugins, supporting automatic code generation, Model Driven Architecture and Development, Transformations, Collaboration, GUI creation, etc. The plugins can be mapped to almost all required functionalities.
- Extensive documentation and support from official sources, organisations, as well as the user community.
4 Generic Service Development Platform Architecture

This chapter discusses the core architecture of the Generic Service Development platform. It is an elaboration of the architecture described in [1] and [22], refined through the requirements set in the previous chapters.

The architectural specification in this document is focused on the components developed within WP42, which do not include the SLM Toolbox, developed within WP15. However, since the SLM Toolbox is an integral part of the development infrastructure, we discuss in depth its integration with the rest of the platform.

The architecture and components described in this chapter represent the final version of the architecture based on the experience from the development of D42.3 “Generic Service Development Platform first prototype”, further research, and collaboration with MSEE stakeholders. The architecture and specifications of the platform have evolved since the initial version. These changes are also documented in this chapter, along with a description of their rationale.

4.1 Architecture

Using the Eclipse platform as a base, the required functionalities are mapped into a coherent framework by selecting and exploiting appropriate plugins and Eclipse-compatible technologies. These plugins and technologies correspond to the majority of the required functionalities of the platform; however, integrating them into a unified and transparent development platform that will support the requirements set in the previous chapters presents a significant development challenge. This will require the creation of custom components and interoperability adapters as necessary (both for internal and external elements), automation, new user interface elements and behaviours, the definition of transformation rules and code templates, user access rules and authentication etc, as well as extensive testing and debugging, in order to finally implement a cohesive development environment.

Excluding the elements of the platform are integrated as shown in the following figure:
Near the top of the figure, the Eclipse platform provides the IDE and a runtime environment that supports the main platform workspace and is responsible for the integration with a set of software development plugins.

The main Eclipse runtime integrates the following elements:
The Eclipse Workbench, which provides the main Eclipse workspace, presenting a ECLIPSE IDE customised for use by MSEE developers through modular plugins

- JFace⁴, a User Interface development toolkit
- The Standard Widget Toolkit (SWT)⁵, a toolkit providing access to the UI features of the target operating systems for applications
- A user workspace, containing user projects i.e. collections of development artefacts

The runtime can integrate a number of development plugins and the following have been selected:

- The Eclipse JAVA development tools
- Plugin Development Environment

The flexibility of the Eclipse platform means that other plugins may be optionally integrated with the runtime environment, such as the Eclipse C/C++ Development tooling and the PHP development tools, depending on user needs. Note that the inclusion of any new technology platform requires the use of a suitable Technology Model to use in the in the TIM to TSM conversion process.

The Eclipse runtime also provides debugging, help/documentation and collaborative development support.

On the right side of the figure, the Generic Development Platform Core is illustrated as a separate entity. The elements of the Core are also integrated with the Eclipse Runtime as Eclipse plugins, and represent functions that provide the central functionalities of the Generic Service Development Platform (i.e. graphical service authoring, model transformations, code generation and the model repository).

The Model Repository Server, which is considered a core component of the platform, is depicted separately, as it is expected to be deployed as a single central repository server for each MSEE platform instance. On the contrary, the other core plugins are executed on the user’s workstation and may be replicated across the MSEE platform (as many developers work simultaneously). Also, instances of the Model Repository Client API and Model Repository View are available to SLM Toolbox users, as well as Mobile Business Platform users.

Some core components are based on existing technologies, but require extensive integration and customisation work in order to conform to the workflow requirements of MSEE. The Core components include the following elements:

- The Service Authoring Environment with model editors for:
  - BPMN (Based on Activiti⁶)
  - UML (Based on Papyrus UML⁷)
  - WSDL (Based on the Eclipse WebTools Platform⁸)
- The Transformations plugin (based on the Acceleo plugin and the MOF Model-to-Text (MTL) language⁹)
- Platform Integration mechanisms

---

⁴ http://wiki.eclipse.org/index.php/JFace
⁵ http://www.eclipse.org/swt/
⁶ http://www.activiti.org/
⁷ http://www.papyrusuml.org
⁸ http://www.eclipse.org/webtools/
⁹ http://www.eclipse.org/acceleo/
The Model Repository stack:
- Model Repository Server (based on WebDAV\(^\text{10}\) as an underlying repository system)
- Model Repository Client API
- Model Repository View
- TIM model import wizard (using the Model Repository Stack)
- Service Development Platform Integration, containing:
  - The Service Discovery wizard
  - The Service Registration wizard

The fact that most major components are based on Eclipse plugins or Eclipse-compatible projects is a direct benefit of the high popularity and support of the Eclipse platform. This allows critical components such as the Transformation Plugin, Service Authoring, Generator and Model Repository components to be directly integrated with the Eclipse Runtime and the Eclipse IDE. This also allows the platform to present a unified environment to users while, generally, reducing the effort required for integration and development.

In a sense, the flexibility of Eclipse as a framework to build entirely custom environments for software development enables the creation of a complete Model-Driven Development environment, specifically tailored for the creation of manufacturing ecosystem services for MSEE.

In the following sections the Service Platform Components and their underlying technologies are examined.

### 4.1.1 Eclipse Runtime

#### 4.1.1.1 Eclipse IDE and Workbench

The Eclipse Workbench is the desktop development environment of the Eclipse programming environment, implementing its UI. It provides access to the plugins and features of the Eclipse platform, including the plugins of the Generic Service Development Platform Core. In order to provide user interface development capabilities, the integration of Jface and SWT is proposed.

The Workbench also provides access to help files, IDE collaboration features and debugging utilities.

#### 4.1.1.2 Development plugins

**Eclipse Java Development Tools (JDT)**

The JDT project provides the tool plug-ins that implements a Java IDE supporting the development of any Java application, including Eclipse plug-ins. It adds a Java project nature and Java perspective to the Eclipse Workbench as well as a number of views, editors, wizards, builders, and code merging and refactoring tools. The JDT project allows Eclipse to be a stand-alone development environment.

**Plug-in Development Environment (PDE)**

The Plug-in Development Environment (PDE) provides tools to create, develop, test, debug, build and deploy Eclipse plug-ins, fragments, features, update sites and RCP products. PDE also provides comprehensive OSGi\(^\text{11}\) tooling, which makes it an ideal environment for component programming in general, not only in Eclipse plug-in development. The PDE is actually used for the development of custom plugins for the Generic Service Development Platform.

---

\(^{10}\) [http://www.webdav.org/](http://www.webdav.org/)

\(^{11}\) [http://www.osgi.org/Main/HomePage](http://www.osgi.org/Main/HomePage)
Platform. The modular nature of Eclipse and the inclusion of the PDE to the actual production environment of MSEE enable developers to develop plugins for use in specific manufacturing service development applications. This way, the Generic Service Development Platform can be extended by its users over time and continuously evolve according to their needs.

Other development plugins
The modular nature of Eclipse allows the inclusion of more development plugins. The initial specification called for the inclusion of C/C++ and PHP tools. It was decided that the platform should include support only for Java (and the PDE). This way, design and development focused on achieving a coherent and reliable development workflow using a single technology model for source code development i.e. Java. However, source code can be generated for any target platform, as long as a suitable technology model (a TSM profile) is defined.

4.1.2 Generic Development Platform Core
4.1.2.1 Model Repository
4.1.2.1.1 Introduction

The Generic Service Development platform relies on a repository module for storing, searching and retrieving models as well as software utilized by both the SLM Toolbox as well as the Development platform itself. This module aims to extend the model access repository as it was initially designed and introduced by the SLM Toolbox Specifications and Design document (Deliverable D15.2) and included in the updated version, D15.3[2].

This module is responsible for the management of access to the model data of the MSEE platform. It provides storing mechanisms and structures for the software development artifacts within MSEE. It offers a set of functional services to the Generic Service Development Platform, as well as the SLMToolbox (SP1) and the MSEE Mobile Business Platform (WP44) in order to store intermediate models while allowing common access. The collaborative, inter-organizational nature of an ecosystem such as within MSEE also imposes the need for privacy, security and data protection; therefore the repository module requires access control and data integrity mechanisms.

The repository stores model and software resources in a centralized location. In the logical architecture of the MSEE IT system, this repository should include one unique commonly accessible access point, which would manage collaborative and concurrent access by multiple SLMToolBox, Service Development Platform, and Mobile Business Platform instances.

4.1.2.1.2 Technology Platform
4.1.2.1.2.1 Methodologies and Technologies Considered

The initial specification of the Generic Service Development Platform called for the use of Eclipse Connected Data Objects (CDO)12 thus extending the model repository defined during the development and early utilization of the modelling environment of the SLMToolBox. CDO is a pure Java model repository and can serve as a persistence and distribution framework for the MSEE GSDev platform.

12 http://www.eclipse.org/cdo/documentation/
However, during the development of the first prototype it became clear that CDO was not suitable for the development of the Generic Service Development Platform prototype. For this reason, a central WebDAV server has been selected as a viable alternative.

WebDAV is an acronym for “Web-based Distributed Authoring and Versioning”. It is a set of extensions for the HTTP/1.1 protocol, adding new methods and headers, request/response formats, etc. These additions provide the ability to lock web resources, associate properties with them (metadata) and namespace management for resource collections. Further features under development include support for ordered collections, versioning and configuration management and secure access control. As such, it provides a powerful tool for storage, authoring, annotation and access of resources by multiple distributed users.[23]

A closer examination of the WebDAV protocol indicates that it is a much better fit for the requirements of the Model Repository. The advantages of WebDAV approach include:

- Implementing a WebDAV-based system is much simpler than a CDO-based repository. WebDAV is based on HTTP-style access to resources, while at the same time, maintains a file-system paradigm for organising shared resources. Also, this simplicity means that it is far easier to build custom wizards and applications on top of it. Finally, using HTTP means that WebDAV can be used whenever standard web access is available, even in corporate intranets with restrictive firewall policies.
- WebDAV has a standard security model based on Access Control Lists. Based on the requirements for the platform and the Model Repository, this security model can be customized to use the general MSEE platform security based on the Federated SSO Utility Service [26]. Also, its reliance on HTTP makes implementing authentication and security features much easier in comparison to other repository platforms (such as SVN). Also, WebDAV features advanced logging capabilities, critical in a security oriented application.
- WebDAV is an open web standard based on HTTP. For this reason, there exist several implementations of the WebDAV protocol, both open source and commercial. Several of those implementations meet the requirements of the Model Repository, enabling flexibility and access to a larger technical support base.
- WebDAV is agnostic with regards to the types of resources, providing only the means to access them over HTTP. On the other hand, CDO only supports storage of ECore-compliant models. For this reason, WebDAV provides significant flexibility to MSEE components and developers in choosing the best model formats for their application. Furthermore, this allows users to store all kinds of development artefacts in the Model Repository, including source code, documentation, binary executables, etc, thus extending its usefulness.
- WebDAV provides capabilities for organising resources in collections and hierarchies, similar to directories in a standard filesystem. Using WebDAV, it is possible to organise models more effectively. For example models can be categorised, by level (BSM, TSM, etc) or by function within the organization (HR, IT, Procurement etc).
- WebDAV natively supports distributed authoring. It includes locking mechanisms for collaborative editing while maintaining information integrity.
- WebDAV natively supports the annotation of resources with properties and metadata. For example, WebDAV can store properties such as the creator of a resource, or the time when it was created or last modified. Of particular interest to the Model Repository is the capacity of WebDAV to extend the default set of resource metadata with custom elements. This way, it is possible to define specific sets of metadata for MSEE models, which, in turn, are used by the MSEE development components.
• WebDAV natively supports search functions through the “Distributed Searching and Locating” (DASL) protocol [24]. Combined with the ability to create custom metadata it makes it possible to create semantically enhanced search features.

4.1.2.1.3 Architecture
The Model Repository is composed of the following components:

• The Model Repository Server, the centralized “single point of truth” repository.
• The Model Repository Client API module, which is delivered as an Eclipse plugin and provides an API via which other Eclipse plugins can communicate with the Model Repository Server.
• The The Model Repository View, which is an Eclipse plugin with a UI, providing the MSEE Model Repository View which allows end users to perform all interaction with the Model Repository Server.

![Model Repository Architecture Diagram]

Figure 5: Model Repository Architecture

The Model Repository Server is deployed as a network server process to a central server. On the other hand, the other two components are executed locally, at the user workstations running the MSEE Eclipse-based tools.

User access to the Model repository is mediated by the two local plugins as illustrated in the figure below:
The Model Repository Server is deployed on a server as a common repository for MSEE service development. Multiple instances of the Model Repository can be supported, e.g. one per ecosystem or per organisation. On the other hand, the two local plugins are present in the multiple instances of the Generic Service Development Platform. Furthermore, the Model Repository View and Model Repository Client are also used by the SLMToolbox to access the Model Repository and upload TIM models during the development process.

4.1.2.1.4 Model Repository Server

The Model Repository Server contains the actual repository, implemented using WebDAV. The Server is implemented as a hierarchical resource repository, with files and folders corresponding to development artefacts and resources. The selected implementation enables users to store, not only models, but any type of file that can be transferred over HTTP.

4.1.2.1.5 Model Repository Client API

The Model Repository Client API functions as an intermediary between the Model Repository Server and the Model Repository view plugin, or any other plugin needing access to the Model Repository.

The API contains a set of methods for CRUD (Create, Retrieve, Update, Delete) operations on a collection of elements organised in hierarchies of folders. The API is used by the Model Repository View plugin and the TIM Import Wizard, as well as the SLM Toolbox.

Basic API operations are listed in the following table (note that IOException are errors reported by the data file system, while DAVexceptions indicate failures of the underlying WebDAV infrastructure):

<table>
<thead>
<tr>
<th>Method Definition</th>
<th>Input</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void configure(ModelRepositoryConfiguration configuration);</td>
<td>• A Model Repository Server Configuration (e.g. host address, port, connections, etc)</td>
<td>Configures the client with the remote repository's configuration (host, port, context path etc).</td>
</tr>
<tr>
<td>void createFolder(String path) throws DavException, IOException;</td>
<td>• Path to folder</td>
<td>Creates a folder at the specified path.</td>
</tr>
<tr>
<td>List&lt;DavResource&gt; listItems(String path) throws IOException, DavException;</td>
<td>• Path to resource</td>
<td>Lists all items which are children of { code path }</td>
</tr>
<tr>
<td>void createFile(String path, File file, String contentType) throws DavException, IOException;</td>
<td>• Path to file • File to stream</td>
<td>Creates a file opening a filestream to the WebDAV</td>
</tr>
<tr>
<td>Type of Contents</td>
<td>server.</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>---------</td>
<td></td>
</tr>
<tr>
<td>Path to file</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Binary Contents</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Type of Contents</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Creates a file with the binary contents provided as parameter.

<table>
<thead>
<tr>
<th>Path to stored resource</th>
<th>Deletes a resource (file or folder).</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Source Path</th>
<th>Destination Path</th>
<th>Overwrite option</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Moves a resource to a new URI. If a resource already exists at the given URI, and the Overwrite parameter is &quot;True&quot; then it is overwritten. Otherwise it fails.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path to resource</th>
<th>Downloads a resource from the repository</th>
</tr>
</thead>
</table>

| None | Provides the root path of the repository (e.g. in 'http://192.168.33.11/repos/", it is "/repos") |

### 4.1.2.1.6 Model Repository View

The Model Repository View provides the main graphical user interface to the Model Repository through the Generic Service Development platform.

The MR View is accessible through the main Generic Service Development platform Graphical User Interface. After initiating a connection to the Model Repository Server, it provides a view of the Model Repository contents as a tree-structure with hierarchical folders and individual resources. Users can navigate the structure and perform operations on it resources, similar to those encountered in traditional file systems.

![Model Repository View](image)

**Figure 7: Model Repository View plugin**

Specifically, the Model Repository View interface allows users to perform the following operations, based on the underlying API and accessible through its GUI:

- Connect to the Model Repository
  - The user provides connection details and options such as host address, ports, etc
• View properties of a specific resource (model) or collection
  o The user can select a model/collection in the tree-view and read its metadata
• Download a resource from the Model Repository
  o The user can select a model in the tree-view and download it to the Eclipse user workspace
• Upload a resource
  o The user can upload a model to the model repository
• Delete a resource or a collection

4.1.2.2 Service Development and Model Transformation Components
The transformation functionality is the “heart” of the Generic Service Development Platform. A number of different methods and technologies for model transformation and code generation have been considered for implementing this function, and were used in developing early prototypes. Subsequent iterations and implementation experiments lead to a robust implementation which has already been used in the first prototype and will be present in the final version of the Generic Service Development Platform.

4.1.2.2.1.1 Methodologies and Technologies considered
The initial specification of the Transformation process anticipated a combination of graphical service authoring, Model-to Model transformation functionalities. TIM-level models would be imported to the Development platform via the Model Repository, and into a graphical service editor based on the Service Component Architecture paradigm. Users would select a Technology Platform, edit these services, reference and import external components, and then use a “Transformation Plugin”, to perform model-to-model transformation to generate TSM-level models. Following the model transformation, a model-to-text component would transform the intermediate TSM model into executable code for the target platform.

4.1.2.2.1.1.1 SCA-based service authoring
The initial specification called for a custom designed graphical editing environment, based on the Graphical Modeling Framework (GMF)\(^{13}\) and the Graphical Editing Framework (GEF)\(^{14}\) of the Eclipse platform. This customised environment was expected to follow the conventions of the Service Component Architecture \(^{17}\) and would be based on the Eclipse SCA Tools project\(^{15}\). The Service Component Architecture (SCA) is a set of specifications which describe a model for building applications and systems.

The SCA Tools would support several of the required features, such as graphical editing, code generator (Java), SAWSDL annotations, etc \(^{19}\)

4.1.2.2.1.1.2 Model-to-Model Transformations and code generation
The initial model transformation features anticipated the use of a set of tools and resources based on the Eclipse Modelling Framework and the ECORE metamodel\(^{16}\), including at least one of the following:
  o The ATLAS Model Management Architecture\(^{17}\) and the ATLAS Transformation Language\(^{18}\)[7][8][9][15]
  o The VIATRA/VIATRA2 (VIsual Automated model TRAnsformations) framework\(^{19}\)[15]

\(^{13}\)http://www.eclipse.org/modeling/gmp/
\(^{14}\)http://www.eclipse.org/gef/
\(^{15}\)http://wiki.eclipse.org/SCA
\(^{16}\)http://www.eclipse.org/modeling/emf/
\(^{17}\)http://wiki.eclipse.org/AMMA
\(^{18}\)http://www.eclipse.org/atl/
An example of the model-to-model process using the initial proposed implementation foundation, (i.e. the Eclipse EMF and the Atlas Transformation language) is illustrated in the following figure (note that the ECORE meta-model is used, as the initial specification called for CDO as the foundation for the Model Repository):

![Model-to-model process diagram](image)

Figure 8: ATL-based model transformation, adapted from [10]

In this context, (using ATL and EMF) the definition of models is performed according to their metamodels, as presented in the figure above. Transformation rules describe how the source metamodel concepts are mapped in the target metamodel concepts. A transformation from a source Model (TIM Model A - TIMa) into a target Model (TSM Model B - TSMb) is therefore conducted by a transformation definition (TIMMa2TSMMb.atl), based on ATL constructs. This transformation definition is also a model [11]. The source and target models as well as the transformation definition must conform to their metamodels (TIM Metamodel A - TIMMa, TSM Metamodel B - TSMMb and ATL, respectively). In addition, the metamodels must conform to a meta-metamodel, in this case the eCORE [12].

4.1.2.2.1.2 Evaluation of the initial approach

Further research and the development of the first prototype demonstrated the need to revise the initial technology and methodology selection.

MSEE Deliverable D11.2 “Service concepts, models and method: Model Driven Service Engineering” [20] (M12) contains of the initial investigation of the MSEE development workflow, from BSM to TSM models. This effort has reached the conclusion that the recommended modelling paradigms at the TIM level are BPMN, UML and WSDL. D11.2 recommends BPMN as a precursor to BPEL models for execution, UML as a framework for component, entities and database design and WSDL for services based on enterprise applications. Furthermore, at the TSM level, deliverable D11.2 recommends the use of BPEL for process executed by enterprise application modules, WSDL for specifying implementation information about web services, UML for further modelling of enterprise applications and internal data/physical schemas for specifying data elements and structure.

Based on these recommendations, work on the service development infrastructures has focused on these specific modelling languages. The SLM Toolbox, as detailed in D15.3[2] is using UML and BPMN as the base of an iterative TIM modelling process. In this process,
abstract representations are reduced to detailed service models using UML, BPMN and internal formats. The final iteration emphasises UML (sequence diagrams and class diagrams). However, in the case of unstable or complex design requirements, the design process may also include parallel modelling of TIM and TSM, meaning that both UML and BPMN input will be expected.

On the other end of the service development process, service registration (as well as service discovery) is managed through the Service Delivery Platform. As stated in deliverable D43.2 “Service Delivery Infrastructure specifications and architecture” [3], the SDP is supports WSDL descriptions and REST services. The first version of the SDP (D43.3) included support for WSDL only, while the final version will also support REST.

For these reasons, the Generic Service Development platform, as an intermediary between these actors, is expected to support UML, BPMN and WSDL. Users of the Development platform will have to import UML and BPMN models and work with WSDL service descriptions. Notably, WSDL descriptions from the Delivery Platform can also support automatic code generation for service clients. These formats are very common and are supported in the Eclipse framework through several open-source plugins.

On the other hand, an SCA-based service authoring environment would include the integration of the following components:

- Transformation logic and import mechanisms supporting UML-to-SCA, BPMN-to-SCA and WSDL-to-SCA
- An SCA service authoring environment adapted to MSEE, including support for TIM technology annotations
- Transformation logic from SCA-to-target source code or SCA-to-intermediate models

The extra components correspond to a significant amount of integration effort, which may not fall entirely within the scope and the goals of the Development platform.

Introducing yet another modelling paradigm would also run the risk of increasing the mental workload of the platform users. Despite the fact that the components of the development workflow (the SLM Toolbox, Mobile Business, Service Development and Service Delivery platforms) are built as separate entities, their users will overlap, and in some cases the same person may access all three components. If SCA were included, users would start with UML and BPMN for the SLM toolbox, would then switch to SCA for composition and enrichment, and would end the process with code editing. Dropping SCA means that users will continue working with UML and BPMN in the composition and enrichment stages as well, without having to switch their mental framework before reaching the code stage.

For these reasons it was decided to focus directly on supporting modelling paradigms already in use in the development workflow, i.e. UML, BPMN and WSDL.

Focusing on UML and BPMN made the use of model-to-model transformation tools unnecessary. Several tools for UML and BPMN already include the capability of annotating a technology-agnostic model with metadata for specific target implementations i.e. the model enrichment process. Additionally, there are several tools that support automatic code generation from UML, BPMN and WSDL models. Therefore, it was decided to not use model-to-model transformations based on either ATLAS or VIATRA2.

4.1.2.2.2 TIM-to-code via the “Model Enrichment” process in the Generic Service Development Platform
The generic transformation flow is illustrated in the following figure, with the dotted rectangles indicating activities and the solid rectangles indicating development artefacts:

![Generic Model Transformation Workflow in MSEE's Generic Service Development Platform](image)

**Figure 9: Generic Model Transformation Workflow in MSEE’s Generic Service Development Platform**

In this flow, the TIM enters the transformation process and:

- A target Technology Model is selected
- The TIM model is edited in the Development Platform’s model editors if necessary
  - Services developed in the Generic Service Development Platform are not required to follow the convention of creating a single “serial” service, following a straight path of interactions and components from beginning to end. Developers are also able to specify single components, collections of components or parallel composites and other topologies.
- The TIM model is annotated with Technology Model- derived semantics
  - Automatically (if available for the model type and target technology)
  - Manually by the service developer
- This “Model Enrichment” process results in the TSM model: a model annotated with Technology-specific semantics and structures
- An automatic code generator produces code for the target platform based on
  - The TSM model annotations and structure
  - The selected technology platform
- References to external services are added to the executable code

Automatic code generation can also be considered as a special case of a model transformation. In this case, a high-level model is converted to a low-level textual representation, corresponding to a specific programming language and target platform. In the MSEE Generic Service Development platform, automatic code generation will be used to convert a high-level service description into finished code or a source code framework, which can then be further elaborated by software developers through editing, testing and debugging. The end result of the process will be an executable build or description of the service.

During this process, external components can be added to the models either directly (like new entities retrieved from a common library) or referenced (as is the case for registered services in the Service Delivery Platform).

The steps of the TIM-to-code process correspond to the selected components and technologies of the final design of the Development platform, as illustrated in the following table:
### Table 6: Development components and the transformation process

<table>
<thead>
<tr>
<th>Action</th>
<th>Component</th>
<th>Underlying Technology</th>
</tr>
</thead>
<tbody>
<tr>
<td>Model Editing (UML)</td>
<td>UML Editor</td>
<td>Papyrus UML</td>
</tr>
<tr>
<td>Model Editing (BPMN)</td>
<td>BPMN Editor</td>
<td>Activiti Designer</td>
</tr>
<tr>
<td>Model Enrichment (UML)</td>
<td>UML Editor</td>
<td>UML Java profile (Papyrus UML)</td>
</tr>
<tr>
<td>Model Enrichment (BPMN)</td>
<td>BPMN Editor</td>
<td>Java and .BAR support (Activiti)</td>
</tr>
<tr>
<td>Referencing External Services</td>
<td>WSDL Editor (supported by Service Delivery Platform integration)</td>
<td>Eclipse Webtools</td>
</tr>
<tr>
<td>Automatic code generation (TSM-to-code)</td>
<td>Transformations Plugin</td>
<td>Acceleo plugin with MOF Model-to-text Language</td>
</tr>
<tr>
<td>Automatic code generation (WSDL-to-code)</td>
<td>WSDL Editor</td>
<td>Eclipse Webtools</td>
</tr>
</tbody>
</table>

The components used in each case correspond to different development workflows depending on the input to the development platform and the desired output. Developers go through the steps of the transformation workflow by using the appropriate components in the integrated Development platform environment. Each of the following workflows is a special case or branch of the generic TIM-to-code process described above.

### UML TIM to executable code

In this workflow, UML class diagrams are converted into Java classes (JavaBeans). It is possible to generate any type of source code given a suitable technology profile. However, in the UML-to-code scenario, the Generic Service Development platform will focus on Java domain model classes and JavaBeans. A sample workflow would involve the following steps:

- The user imports the UML Java profile (a Technology Model) from the model repository using the TIM Models Import Wizard
- The user imports a TIM model from the model repository using the TIM Models Import Wizard
- The user views and edits the TIM model in the graphical UML Editor (Papyrus UML)
- The user requests the application of the imported UML Java Profile to the TIM model
- The Java profile is applied and the UML model is automatically prepared for manual annotation (e.g. new interface elements allow users to assign Java data types to entity attributes)
- The user manually annotates the model in the graphical UML editor with semantics derived from the Java profile. For example:
  - UML entities are marked as Java beans
  - Entity attributes are assigned Java data types (such as float, integer, date, etc)
- After completing the enrichment process, the user requests the generation of source code.
- The platform generates a Java code template, based on the enriched model (the TSM model). The template contains empty Java classes (corresponding to entities) with the selected attributes, methods, etc.
- The user works on the resulting source code using standard software development techniques

This workflow can be used to generate source code components (e.g. Java beans) or a complete service which is then registered with the service development platform.
registration process includes the deployment of the service, the generation and semantic annotation of a WSDL description and use of the Delivery Service Integration functions to register the resulting SAWSDL description. The registration process is described in further detail in the “Integration with other MSEE Platform Components” section.

The figure below illustrates the Generic Service Development Platform GUI during the UML model enrichment process as a special case of the generic transformation process:

![Figure 10: UML-to-code transformation workflow](image)

Adding references to registered services: WSDL service description to executable code
During development, users are expected to add references to services registered with the Service Delivery Platform. Services are discovered and imported as WSDL through the Service Delivery Platform integration functionality of the Development platform. This workflow involves the following steps:

- A WSDL service description is imported using the service discovery and importation functions of the Development platform
- The WSDL description is loaded into the WSDL Editor (Eclipse Webtools)
- The user edits the WSDL description if necessary

![Figure 11: UML-based TSM Model with Java bean stereotyping and data types](image)
The user requests the generation of executable code for a client using the specific web service

The platform generates the client code (a web service stub based on Java API for XML Web Services/JAX-WS\(^{20}\))

The users can then work on the client source code and add it directly to the source code generated via the UML-to-code process. The process is illustrated below as a branch of the generic flow:

![Figure 12: WSDL-to-code transformation workflow](http://docs.oracle.com/javaee/5/tutorial/doc/bnayl.html)

**BPMN TIM to executable Business Archive (.BAR) File**

BPMN input is used to generate executable service orchestrations as BAR files. The resulting Service Deployment Artefacts can then be executed by MSEE’s Innovation Ecosystem Platform. This workflow involves the following steps:

- The user imports a BPMN TIM model from the Model Repository
- The user loads the BPMN model into the BPMN Editor (Activiti)
- The user enriches the model description in the Activiti Designer environment by adding semantics relevant to execution by the IEP engine. Execution metadata may include, among others:
  - Variables
  - Forms
  - Java source code
  - Scripts
- The user requests generation of the executable .BAR file
- The platform generates an executable .BAR file based on the enriched BPMN model

This process is illustrated in the following diagram as a special case of the generic transformation process:
The platform components supporting this process are further described in the following sections.

### 4.1.2.2.3 Implementation of the Service Authoring Environment

The specification of the Generic Service Development Platform calls for the provision of means to edit and manipulate models of services at the TIM and TSM levels. The Platform includes a set of editors allowing users to work with UML, BPMN and WSDL descriptions. These are accessible as parts of the integrated development environment. From a technical perspective, the editors are customised Eclipse plugins, based on off-the-shelf components.

#### 4.1.2.2.3.1 UML Editor

The UML Editor is based on the Papyrus plugin\(^{21}\) for Eclipse. Papyrus is a graphical editor for Eclipse Modelling Framework models, but mainly UML2.4.1, (focused on compliance with the OMG specification of UML 2\(^{22}\)). Papyrus has been released under an open source license, \(^{21}\)http://www.papyrusuml.org/  
\(^{22}\)http://www.uml.org/

The Papyrus plugin allows for extensive customisation, a very important feature in relation to the need to adapt it to the needs of MSEE and its users. Papyrus provides support for UML profiles, which are sets of specifications, used for customising “general purpose” UML and creating domain-specific versions of the language. This allows the definition of domain-appropriate terminology, syntax and notation conventions, as well as new semantics, constraints and mapping rules [27].

Using UML Profiles it is possible to create a custom environment for the transition from TIM-level UML to TSM-level, and then to source code. This is achieved by modifying the editing environment with the application of the appropriate profile. In the Generic Service Development Platform, the UML JAVA Profile is used with Papyrus.

Additionally, UML profiles can be used to customise the editing environment according with features deriving from the MSEE platform, the manufacturing domain, a particular ecosystem or organisation. Finally, since the plugin is released as open source, it’s possible to further customise it according to the particular needs of the MSEE platform.
4.1.2.2.3.2 BPMN Editor
The BPMN editor of the Generic Service Development Platform is based on the Activiti Business Process Management Platform. Activiti is focused on manipulating and executing BPMN2 process models, while also providing business-oriented features such as business intelligence on executed process, logging, etc.

The Activiti platform components are organised in three separate classes:

- Modelling, including Activiti Modeler, Activiti Designer and Activiti Kickstart
- Runtime, including the Activiti Engine
- Management, including Activiti Explorer and Activiti REST

The BPMN editor is based on the open-source modelling component, and, specifically, the Activiti Designer Eclipse plugin. The plugin provides functionality for creating and editing BPMN models, as well as annotating them with specific execution information. Also, the Activiti designer can generate “Service Deployment Artefacts”, i.e. executable .BAR files. The Activiti designer can be further customised by modifying its user interface, adding new custom modelling elements, implementing logic for model validation, etc.[28]
4.1.2.2.3 WSDL Editor

The WSDL Editor of the Generic Service Development Platform is based on the open-source Eclipse Web Tools Platform (WTP) project. The WTP provides tools for the development of Web and Java Enterprise Edition applications in the Eclipse framework [29]. In the Generic Service Development platform, the WTP is used for editing WSDL service descriptions, annotating WSDL (producing SAWSDL), and generating source code for a client of a specific web service based on its WSDL description.

4.1.2.2.4 Implementation of the Transformations Plugin

The Transformations plugin is based on the Acceleo Eclipse plugin. The Acceleo plugin is based on the MOF Model To Text Transformation Language[30][31]. The Transformations plugin can automatically produce source code from an annotated UML model (a TSM model) and make it available to the project workspace.
4.2 Integration with other MSEE Platform Components

The Generic Service Development Platform will interoperate with a set of MSEE components. These are:

- The SLM Toolbox
- The Service Delivery Platform
- The Mobile Business Platform

Interoperability with the rest of the MSEE platform is handled by the MSEE Generic Service Development Platform Plugin. The specification is a result of prototype development and close cooperation with efforts in WP43, WP44, and WP15.

4.2.1 Integrated service development workflow

The Generic Service Development platform participates in the service development process by interoperating with the SLM Toolbox and the Service Delivery Platform.

The integration between the SLM Toolbox and the MSEE Generic Service Development Platform is mediated by the Model Repository. The Model Repository is the primary storage area for registering, searching and retrieving models created and transformed by the SLM Toolbox and the MSEE Generic Service Development Platform.

The integrated flow focuses primarily on the transformation of TIM models into TSM models (TIM-into-TSM). Within the scope of the integration, transformation is a refinement of models that incorporates details of a specific platform to the source TIM model. In a model refinement, most elements from the source model (TIM) are copied to the target model (TSM), while other elements must be changed in order to incorporate platform and technology-specific aspects.

Source code is then generated, the service is deployed and the process ends with the registration of Generic Service Development Platform products with the SDP.

The figure below portrays, at a high level, the fully integrated workflow for service development within MSEE. It features interactions between the SLM Toolbox, MSEE Generic Service Development Platform and MSEE Service Delivery Platform.
4.2.2 Integration wizards

Integration with other MSEE platform actors is mainly handled through a set of modules and functionalities that enable the Generic Service Development Platform to exchange development artefacts with other MSEE platform actors.

The Import/Export Wizard plugins:
- Search, discovery and importation of elements registered with the Service Delivery Platform.
- Importation of TIM models to the Generic Service Development platform via the Model Repository.
- Registration of services with the Service Delivery Platform.
4.2.2.1 Integration with the SLM Toolbox

4.2.2.2 Introduction

The SLM Toolbox is primarily responsible for providing a set of tools for three different aspects:
(a) Modelling
(b) Service engineering, and
(c) Monitoring and control.

The SLM Toolbox is handling the initial stages of modelling in the proposed MSEE Model Driven Architecture (MDA) environment (i.e. BSM and TIM), whilst the MSEE Generic Service Development Platform is handling the transformation from the TIM model into the TSM metamodel as well as all consecutive steps of authoring a service, generating the respective source code and deploying the final outcome into the MSEE Generic Service Delivery Platform.

The expected input from the SLM Toolbox can be better understood in the context of the transitions between the three modelling levels (BSM, TSM, TIM) involved in the service development process.

These levels are illustrated in the following figure:

![Diagram](image)

**Figure 19: BSM, TIM and TSM modelling levels**

In the upper transformation level, the BSM is considered to be a descriptive model called “Requirements Model”. This is a detailed collection of use cases and activity diagrams. These are manually mapped to a TIM model, containing the following elements:

- **Dependency Model**: This model describes all components dependencies using model language (e.g. UML/UML2). Class diagrams provide detailed information on classes’ skeletons and their functionalities.
- **Architectural Realisations**: A series of architectural type sequence diagrams that gets input from the Dependency division Model and complies with UML standards. The Dependency Model and the Architectural Realisations are kept consistent throughout the modelling phase.
- **Logical Data Model**: This model describes the data involved in the application under development. The Model holds the information as class diagrams.
The **Schema Definition** is a database schema created through manual coding and based on the Logical Data Model. It provides definitions regarding database structure, tables, columns, fields, owner user name, encoding, collation, grants and access right credentials. The definition of the database schema represents an intermediate stage between the TIM and TSM levels, however it is an artefact created and manipulated at the TSM level, and not expected to be supported by the SLM Toolbox.

Creating the TSM elements and source code from the TSM model employs a combination of manual coding and automatic code generation within the Generic Service Development platform as illustrated in the previous figure.

### 4.2.2.3 Interoperability specifications

Communication between the SLM Toolbox and the Generic Service Development is achieved through shared access to the Model repository. From the perspective of the Generic Service Development platform, this interconnection is mediated by the TIM Models Import Wizard.

The TIM Models Import Wizard provides a graphical interface providing quick access to the Model Repository. It is directly accessible from the Eclipse Development environment and utilises the Model Repository Client API plugin to connect to the Model Repository Server.

Note that since the WebDAV implementation of the Model Repository Server allows the storage of many types of files and resources, the wizard can also be used to import various resources in the Development platform, such as UML profiles, etc.

The usage of the TIM Models Import Wizard is demonstrated in the following figures.

1. The TIM Models Import Wizard is selected from the Eclipse “Import” menu:

   ![Figure 20: Running the TIM Models Import Wizard](image)

2. The user initiates a connection to the Model Repository:
3. The user selects the models to import into the development project

Figure 22: Selecting TIM models to import

4. The user selects the importation destination
Once the user selects “Finish”, the models are imported to the user’s project in the Development platform IDE.

### 4.2.2.4 Integration with the Service Delivery Platform

The Service Delivery Platform (SDP) handles discovery, ranking and selection, grounding and invocation, and monitoring of services that originate from the Generic Service Development platform.

Communication between the two platforms is bi-directional and occurs in two cases:

- During the registration of a new web service to the Service Delivery platform
- During discovery and retrieval of WSDL service descriptions by the Generic Service Development platform during the development process

The Service Delivery Platform Integration plugin covers both registration and discovery of services.

#### 4.2.2.4.1 Service Deployment and Registration

The two platforms communicate after the final stages of the transformation process, where a standalone application or a web service is registered with the SDP’s Registry module.

After generating service source code, the development process in the Generic Service Development Platform enables users to develop services based on that code, using typical software engineering processes (editing, debugging, building binaries, etc). Despite being integral to the MSEE service development process, these activities are typical for the software engineering domain and are beyond the scope of MSEE.

However, the registration of the resulting web service is integral to the MSEE service development workflow. This is executed as follows:

- The web service is deployed normally (to a server) using WebTools
- A WSDL description is generated automatically
- The user adds semantic annotations to the WSDL, producing a SAWSDL description
The SAWSDL description is the end product that needs to be registered with the Service Delivery Platform. To achieve this, the Service Delivery Platform Integration plugin invokes the Registration API of the Service Delivery Platform [3], and uploads the service description to it.

4.2.2.4.2 Service Discovery

The Generic Service Development platform also provides functionalities enabling the search and discovery of services registered with the SDP’s Registry module in order to compose them or connect them to other services during the development process. The Development platform takes advantage of the Discovery API [3] in order to retrieve WSDL service descriptions and use them in the development process.

The service discovery functionality is implemented as a graphical wizard with the following steps:

1. The user invokes the service discovery wizard from the Development platform IDE
2. The user selects search categories from a pre-populated list with available service categories, based on the annotations and metadata of the registered services:

   ![Figure 24: Searching for service categories](image)

3. The user selects a category and the wizard returns the available services
4. The user then selects the desired service, and clicks ‘Finish’
5. A WSDL description of the service is imported to the Development IDE.

4.2.3 Integration with the Mobile Business Platform
The MSEE’s Mobile Business Platform [4] adds implementation and support and support for exposing these resources via a REST API and consuming these resources on mobile clients. As the Generic Service Development Platform is used in the TIM transformation process, it creates a set of specifications for software components in an XML meta-data interchange format (XMI).

This meta-information is collected by the Mobile Development platform, a sub-component of the Mobile Business Platform, able to produce programming code suitable for mobile devices (devices running Mobile Operating Systems, such as Android). The Mobile Development Module extracts a model of a REST resource developed within the Generic Service Development Platform and uses it to generate the necessary code and facilitate the development of a mobile-specific web-service and its mobile client.

Thus, the Mobile Development module of the Business Mobile Platform extends the functionalities of the Generic Service Development platform by adding the ability to produce compatible desktop/web services or applications for mobile devices [4].

4.3 Further work towards the final prototype (M30)
The final prototype will include specific design enhancements which are the result of prototype testing and evaluation, as well as the expected evolution of other interoperating MSEE platform components.

4.3.1 Model Repository

4.3.1.1 Model Repository Support for Collaboration and Security
The Model Repository is expected to be a common resource for MSEE developers working on different services, in various organisations, virtual manufacturing enterprises and manufacturing ecosystems. The main goals of an effective collaborative work scheme are:

- Ensure that users can collaborate by working on the same project artefacts without conflicts and consistency errors.
- Ensure that users can access only the artefacts which are relevant to their project, and are consistent with their level of business security clearance

This means that the Model Repository needs to enforce

- Locking and consistency mechanisms on commonly accessible resources
- A graduated access control scheme distinguishing user privileges according to user, development team, manufacturing ecosystem entity (e.g. a company) and ecosystem.

In order to implement the first feature, the Model Repository in the final version of the Generic Service Development Platform will make use of the locking and consistency mechanisms provided by WebDAV. The resource locking mechanism will allow exclusive editing of models and enhance collaboration between multiple concurrent users.

For the second feature (security) the model Repository will need to interoperate with MSEE’s established access control function, i.e. the Federated Single Sign-On Utility Service developed in WP33 (SSO Utility Service) [26]. Implementing the SSO functionality will require updating and extending the implementation of all three Model Repository Components (Server, Client API and View). Also, the final implementation of this feature is dependent on successful interoperability with the SSO component.

Using the SSO Utility Service the Model Repository will provide federated identity management & access control to models according to each user's role in the ecosystem. For example, business analysts would be able to access BSM and TIM models for their project and workgroup, and developers would access only the corresponding TIM models.

4.3.2 Generic Development Platform Core

4.3.2.1 Service Authoring environment

While the authoring capabilities in the first prototype have been demonstrated, there is significant room for improvement with respect to ease-of use and the particular needs of MSEE developers. For this reason, a number of enhancements to the editing tools and processes can be proposed, which are envisioned to significantly improve user experience and developer productivity in the MSEE context.

MSEE Service composition in TSM BPMN designer

MSEE developers are expected to work within the context of specific applications, organizations, or ecosystems. A developer working in this way will frequently need to re-use existing services developed for each particular domain.

The interface of the ACTIVITI-based BPMN plugin can be enhanced by the addition of a view of REST and WSDL services already registered with the Service Delivery Platform. This “discovery view” could contain collections of service categories for in the SDP, which users will be able to navigate.

This will be complemented by the creation of a custom node type in the Activiti BPMN editor for ‘MSEE service invocation’. Dragging a service from the Service Delivery Platform
discovery view to such a node will associate the node with invocation of the selected service. Additional properties tabs or wizard pages will allow mapping of process variables to service invocation parameters.

Implementation of this feature requires modification of the existing BPMN Editor plugin, as well as successful integration with the Service Delivery Platform (WP43) and the Innovation Ecosystem Platform (WP26).

**Ontology browser view for SA-WSDL annotation**

The first prototype requires developers to annotate WSDL descriptions by directly editing the relevant XML documents. This process can be greatly improved with the implementation of an ontology browser integrated in Service Development Platform which will be able to lookup entities in an ontology. In the case of MSEE, where developers will work in well-defined manufacturing application or ecosystem contexts, the semantic information could also be organised into domain-specific collections and hierarchies.

Developers could use this view to easily locate the appropriate entities either by navigating through a hierarchical view or by using a search function. Once located, semantic information could then be dragged them into the WSDL editor interface and dropped on the relevant service element. This annotation method would reduce overheads in annotation tasks and promote re-use of development assets. (An implementation of a similar “drag-and-drop” annotation feature is the Ontology Concept Digger\(^{23}\) included in the Eclipse SCA tools).

The implementation of this feature depends on referencing an existing ontology for SA-WSDL annotations.

**4.3.3 Integration with other MSEE components**

**Integrated deployment of BPMN artefacts to IEP**

BPMN models received as TIM input lead to the development of .BAR files, which are executable by the Innovation Ecosystem Platform. The design of both platforms does not include a direct means of integration: following the “stand-alone” artefact use case extension described in the use case analysis, the resulting .BAR files are saved in the user’s workspace and are uploaded at a later stage to the Innovation Ecosystem platform.

This process is inefficient and does not conform to the vision of the MSEE platform as a fully integrated software platform. Therefore, a way to integrate the IEP and the Development Platform will be investigated and implemented in cooperation with WP26 stakeholders. This process will allow users to directly deploy developed BPMN processes (as .BAR files) to the IEP from within the Development platform environment. The integrated deployment feature will be developed in cooperation with WP26 stakeholders.

**Support for REST service registration & discovery via Service Delivery Platform**

The Service Delivery Platform design and specification anticipates support for the registration of both WSDL service descriptions (following the SOAP spec) as well as REST (Representational State Transfer) web services. The Generic Service Development platform is designed to use WSDL descriptions both for registration and discovery. On the other hand, REST descriptions are mainly used by the Mobile Business Platform for development of mobile applications.

While WSDL (and the associated SOAP paradigm) is sufficient for the purposes of the Development platform, it can be enhanced to also support work with REST services. With this upgrade, the final version of the platform will be able to provide equivalent functionality of service discovery and client generation as it already does for SOAP/WSDL services. The design and implementation of this feature will be performed in cooperation with WP43 stakeholders developing and designing the Service Delivery Platform.

4.3.4 General User Interface improvements

The Development platform presents a unified environment to its users by using custom integration and plugins, as well as off-the-shelf components. All functions are accessible through a single interface, while wizards have been developed to streamline the execution of sequential activities.

However, there is still room for improvement and further streamlining of the user interface. Wherever applicable, new wizards and graphical editors will be investigated and introduced to guide users through the service development process.
5 Conclusions

This deliverable presented the current state of research on the specifications and architecture of MSEE’s Generic Service Development Platform. The Generic Service Development Platform focuses on supporting the implementation stage of the service life cycle, by enabling the transformation of Technology-Independent Models to Technology-Specific models and creating executable application and service code.

The main flow of operations and a number of extensions have been studied and this analysis resulted in the provision of a set of requirements for the implementation of the platform, while also a set of potential existing candidate environments for the technology foundation of the platform in relation to these requirements have been examined.

Through this process, the Eclipse environment has been selected as the best candidate for the platform, by virtue of its extensive support for customisation, the large number of plugins corresponding to the required functionalities and the large amount of support and documentation. Based on this selection, an attempt to map the requirements for the functional modules of the platform to specific Eclipse-compatible technologies and implementations has been performed. The results of this effort indicate that the adoption of this technology foundation is correct and provides the opportunity to tailor the Generic Service Development platform to the needs of MSEE and, potentially, as a reference implementation for service development in the manufacturing domain.

This specification is a refinement and extension of the initial specification presented in deliverable D42.1. This update is based on the experience gained from the creation of the prototype (Deliverable D42.3 “Generic Service Development Platform first prototype”) in M18 and further research. Development efforts started with the concepts of the initial design and applied them towards the creation of the prototype.

During this process, various potential points of improvement came to light. These involved modifications of the initial requirements and workflows, adjustments of the platform architecture and changes in the technologies and tools used. Furthermore, through cooperation with other MSEE stakeholders, the design and development process took into account the evolution of other interconnected components. Specifically, work in WP42 took place in close coordination with efforts in WP43 (Service Delivery Platform), WP44 (Mobile Business Platform), and WP15 (Service Lifecycle Management Tool), while also taking considering use case needs and the evolution of the project as a whole. As partners in the other Work Packages undertook a similar process of design and prototype development, the new functionalities and interfaces required similar adjustments in the design of the Generic Service Development Platform.

The result of this process on the design level is documented in this deliverable. The evolution of the initial design is documented and discussed in depth in this document. Furthermore, a set of proposed technical improvements over the first prototype has been defined, with the intention of enhancing user experience, productivity and platform integration.

The updated specifications in this document represent the final concept for the design of the Generic Service Development platform, which will be used for the implementation of deliverable D42.4 “Generic Service Development Platform first prototype” in M30 of this project.

The technical evolution so far and the further work planned for the Generic Service Development platform are summarised below:
Activities to date

M12 – Initial specification: D42.1 “Generic Service Development Platform specifications and architecture” (Initial Specification)

M18 – First Prototype of the Generic Service Development Platform (D42.3) including:
   a. Customised development IDE based on the Eclipse IDE
   b. Model Repository including API layer and Graphical User Interface Plugin
   c. UML Editor
   d. BPMN Editor
   e. WSDL Editor and tools
   f. Code Generation (UML to code, BPMN to executable business archive, WSDL to client code)
   g. Service Delivery Platform Integration (Service Discovery and Registration)

M19-M21: Evaluation of the first prototype, design updates and further development towards the final prototype

M21: Updated specification: D42.2 “Generic Service Development platform specifications and architecture” (this document)

Future Activities

M21-M24: Development of intermediate version, focusing on enriching the BPMN editor with service palettes populated with Service Delivery Platform resources.


M24-M30: Development of the final prototype, including:
   a. Improvements to the Model Repository stack components for the addition of access control using the Federated SSO Utility Service (WP33) and resource locking
   b. Other updates to the Model Repository functionality in coordination with other platform stakeholders (esp. WP15, WP43 and WP44).
   c. Implementation of service authoring interface updates
   d. Implementation of interoperability with the Innovation Ecosystem Platform (in coordination with WP26 partners)
   e. Implementation of development workflow for REST services
   f. Further improvements to the development workflow and the user interface
   g. Updates to interoperability mechanisms in coordination with other platform stakeholders (esp. WP15, WP43, WP44 and WP26).

M30- Final prototype of the Generic Service Development Platform (D42.4) including:
   a. Fine grained access control to Model Repository resources
   b. Resource locking and improved collaborative work with Model Repository resources
   c. Updated WSDL editor interface with drag and drop features
   d. Integration with the Innovation Ecosystem Platform from within the Generic Service Development platform IDE
   e. Support for REST service registration and discovery, and other REST-oriented workflow features
   f. Other enhancements to interoperability, reliability and user experience according to the updated specification, use case needs and work performed in parallel on related MSEE components (in Work Packages 43,44,15,33 and 26).
References

[1] MSEE Deliverable D41.1: "MSEE Service-System Functional and Modular Architecture"
[3] MSEE Deliverable D43.2: "Service Delivery Infrastructure specifications and architecture"
[4] MSEE Deliverable D44.2: "Mobile Business platform specifications and architecture"
[22] MSEE Deliverable D42.1: "Generic Service Development platform specifications and architecture”
Appendix

State of the art analysis of visual programming environments and evaluation (D42,1 – M12)

The programming environments examined in this section are:
- JDeveloper
- Geany
- IntelliJ IDEA
- Netbeans
- Eclipse

These environments have been selected as potential candidates for the technology foundation, as they are general development graphical IDEs, enjoy a fair amount of support and are freely available.

For each environment, the main functional characteristics are examined and a brief evaluation of its potential relevance to the Generic Service Development platform is provided. This evaluation was based on the following criteria:
- Functionalities (native or as plugins) relevant to the purposes of the Generic Service Development Platform
- The presence of collaborative development features
- Support and documentation, either from the originating organisation/developers or through a user community

JDeveloper

Oracle's JDeveloper24 is a freeware integrated development environment, suitable for building service oriented applications. It supports well-known industry standards for Java, XML, Web services and SQL. JDeveloper includes features for modelling, coding, debugging, testing, profiling, tuning and deploying applications.

The IDE platform of JDeveloper is licensed under Oracle’s proprietary license (OTN)25. JDeveloper introduces the Oracle’s Application Development Framework (ADF), which simplifies development by minimizing the need to write code that implements design patterns and application’s infrastructure. The ADF splits the application’s architecture in Model, View, Controller and Business Services layers. The last layer includes Universal Modelling Language (UML) modellers for modelling and creating objects, classes and business components. Developers can also use drag and drop techniques to design databases and then implement a programming language (Java) interface to access them.

Using the Model Layer of the JDeveloper IDE, components can be easily bound from the Business Service layer to the Controller and View layers using drag-and-drop methods. JDeveloper also includes a standard debugger, a graphical GUI Builder, autocomplete features and via an extra plugin, version control system (VCS) support.

Evaluation

A critical part of the Generic Service Development Platform is the ability of the IDE platform to support UML modelling techniques and Model Driven Architecture. Classes, use cases, activity and sequence modellers are also required for additional development scenarios. Sequence diagrams can be generated through reverse engineering of classes into diagrams or vice versa. The platform includes some plugins that might be relevant to the Generic Service.

---

Development such as Model Driven Architecture support. However, it has limited support for automated code generation or transformations from business artefacts to executable code.

The platform supports collaborative development via the Oracle Team Productivity Center\textsuperscript{26}, which is also available under Oracle’s OTN license.

JDeveloper is a quite popular platform and has extensive documentation\textsuperscript{27}.

**Geany**

The open source Geany IDE platform\textsuperscript{28} is a light-weight and free suite, available via an open source GPL license. It includes syntax highlights, symbol name auto-completion, multiple platform and multiple programming languages support, symbol lists, code navigation and built systems to compile and execute code. The lack of a dedicated GUI Builder might be a disadvantage, making the platform light-weight, though a debugger and version control support are available with the installation of the relevant plugins.

**Evaluation**

The Geany IDE has generally limited features. Regarding the Generic Service Development platform, the IDE does not directly support Model Driven Development and automated code generation. Geany also relies on the GIMP Toolkit\textsuperscript{29} libraries, which facilitates deployment of graphical user interface applications to multiple platforms (CPUs and Operating Systems).

Geany does not support collaborative development natively.

Geany's documentation is short and support is relatively limited, although developer-supplied manuals\textsuperscript{30} and a community wiki\textsuperscript{31} are available.

### IntelliJ IDEA

IntelliJ IDEA\textsuperscript{32} is a Java Integrated Development Environment, and includes UML-like class diagrams and visual modelling. Also, it supports the Play Framework\textsuperscript{33}, dependency and data flow analysis, Apache Maven\textsuperscript{34} and an abundance of other features and technologies.

The platform includes a live debugger, a GUI builder, version control System and auto complete functions by design. It also installs on multiple platforms and CPU architectures. The IntelliJ IDEA Community Edition is free and open source.

**Evaluation**

The IntelliJ IDEA platform lacks support for certain functionalities in the Generic Service Development platform. It can’t implement code transformations based on TIM - TSM modelling flows and it has limited support for Model Driven architectures (Mostly UML and java code generation through the Smart Development Environment plugin\textsuperscript{35}). Therefore, it appears relatively inconsistent with the requirements of the Generic Service Development Platform.

\textsuperscript{28} [http://www.geany.org/](http://www.geany.org/)
\textsuperscript{29} [http://www.gtk.org/](http://www.gtk.org/)
\textsuperscript{31} [http://wiki.geany.org/](http://wiki.geany.org/)
\textsuperscript{32} [http://www.jetbrains.com/idea/](http://www.jetbrains.com/idea/)
\textsuperscript{33} [http://www.playframework.org/documentation/2.0.4/Home](http://www.playframework.org/documentation/2.0.4/Home)
\textsuperscript{34} [http://maven.apache.org/](http://maven.apache.org/)
\textsuperscript{35} [http://plugins.intellij.net/plugin?pr=idea&pluginId=242](http://plugins.intellij.net/plugin?pr=idea&pluginId=242)
IntelliJ IDEA can support team-based development, through the Team City\(^{36}\) software and can support agile development via YouTrack\(^{37}\). Both products are developed by JetBrains (the IntelliJ IDEA originator organisation).

The platform enjoys adequate support, with documentation\(^{38}\) and developer community resources\(^{39}\).

**Aptana**

The Aptana IDE\(^{40}\) is multi-architecture, multi operating system compliant. It can run in both 32 and 64 bit processing unit architectures. It can also be installed in Windows, Linux and other operating system platforms. Licensed under the GPL proprietary license, it combines several common features of IDEs such as debugger, auto complete functions, GUI builder and version control support (by using an appropriate plug-in).

**Evaluation**

The Aptana IDE includes several JavaScript libraries, with jquery\(^{41}\) being the most popular one. However, it lacks certain features that could make it compatible with the Generic Service Development platform. It lacks Model Driven Architecture support, model transformation, and relevant plugins that could initiate automated code transformation and automated code generation. Aptana appears to be suitable mostly for web-based applications.

Aptana provides support for collaborative development via integration with Git\(^{42}\). The IDE enjoys some community support\(^{43}\) and developer-provided documentation\(^{44}\).

**NetBeans**

NetBeans\(^{45}\) is an open source, cross platform, multi architecture IDE. Some of the core features include user interface management (e.g. menus and toolbars), user settings management, storage management (saving and loading any kind of data), window management, wizard framework (supports step-by-step dialogs), NetBeans Visual Library and Integrated development tools. Released under Oracle’s Common Development and Distribution Licence, the Visual Programming Environment of this IDE supports auto completion, a debugger, a GUI builder and a Version Control System.

The NetBeans Profiler is a module tool for monitoring Java applications tracing down memory leaks and making code optimization an easy to accomplish goal. The GUI design tool allows developers to use swing GUIs\(^{46}\), enabling easy graphical elements add and design on the application under development, without excessive amounts of code manipulation to accomplish that same result.

**Evaluation**

Model Driven Architecture technologies, automatic code generation support and TIM-TSM transformations are not currently in the NetBeans’ active feature list, making it incompatible with the Generic Service Development platform’s core requirements.

\(^{36}\) http://www.jetbrains.com/teamcity/
\(^{37}\) http://www.jetbrains.com/youtrack/index.jsp
\(^{38}\) http://www.jetbrains.com/support/idea/index.html
\(^{39}\) http://devnet.jetbrains.net/community/idea
\(^{40}\) http://www.aptana.com/
\(^{41}\) http://jquery.com/
\(^{42}\) https://wiki.appcelerator.org/display/tis/Git
\(^{43}\) http://stackoverflow.com/questions/tagged/aptana
\(^{44}\) https://wiki.appcelerator.org/display/tis/Home
\(^{45}\) http://netbeans.org/
\(^{46}\) http://en.wikipedia.org/wiki/Swing_(Java)
The IDE natively supports collaborative development via its “Connected Developer” feature\(^\text{47}\), including an issue tracker, version control, developer messaging and collaboration, as well as integration with home.java.net.

Netbeans is supported by an extensive knowledge base\(^\text{48}\) and an active community\(^\text{49}\).

### Eclipse

A well-known open source integrated development environment (IDE), the Eclipse framework platform provides an environment to develop applications in various programming languages and also supports automatic code generation. Eclipse is a multi-language and multi-platform software development environment and includes an extensible plug-in system. Auto complete function, debugger, GUI Builder and version control support are standard features of the Eclipse IDE. Licensed under GPL open source license, the IDE is free, no cost to use and extend among several IDEs available.

Eclipse’s architecture follows a multi-platform paradigm. Several platforms are available to software developers with which they can carry out difficult and demanding software development tasks. The Rich Client Platform helps developing general purpose applications, the Server Platform provides support for remote debugging, variable watch on application servers, such as Tomcat or Glassfish and the Web Tools Platform is a collection of tools for developing Web and Java EE applications.

The Modelling Platform is perhaps the most important feature with respect to the goals of MSEE’s Generic Service Development platform. The Modelling Platform is customisable with various components compatible with the Eclipse Modelling Framework (EMF). The most important features are model transformation, model development tools, concrete syntax development, abstract syntax development, as well as integration between the available modelling tools.

### Evaluation

The Eclipse Platform\(^\text{50,51}\) uses plug-ins to provide all functionality within and on top of the runtime system, in contrast to some other applications, in which functionality is hard coded. A significant sub-division of the Eclipse Platform, which is most relevant to the Generic Service Development platform, is the Modelling project, a container of all the official projects of the Eclipse Foundation focusing on model-based development technologies. Based on the Eclipse Modelling Framework, these components provide several features that correspond to most of the required Functionalities for Model Driven Development in the Generic Service Development Platform.

The IDE supports collaborative development via a range of plugins. At the time of writing, there exist 30 individual plugins\(^\text{52}\) supporting collaborative development with the Eclipse platform.

Eclipse has extensive documentation\(^\text{53}\), an official wiki\(^\text{54}\) is very well supported by an extensive community\(^\text{55}\).

\(^{47}\) http://netbeans.org/features/ide/collaboration.html
\(^{48}\) http://netbeans.org/kb/index.html
\(^{49}\) http://netbeans.org/community/index.html
\(^{50}\) http://www.eclipse.org
\(^{51}\) http://en.wikipedia.org/wiki/Eclipse_(software)
\(^{52}\) http://marketplace.eclipse.org/category/categories/collaboration
\(^{53}\) http://www.eclipse.org/documentation/
\(^{54}\) http://wiki.eclipse.org/Main_Page
**Conclusions**

In this section, a description the operating environment of every part of the MSEE Service Development Platform is provided in two steps, using the analysis of the previous chapter. Initially, the existing technologies are compared with respect to their License, le OS/platform support and features like debuggers, Graphical User Interface builder and version control features. This comparison is summarised in the table below:

<table>
<thead>
<tr>
<th>IDE</th>
<th>Developer</th>
<th>Latest Stable Release</th>
<th>Platform</th>
<th>License</th>
<th>Debugger</th>
<th>GUI Builder</th>
<th>VCSSupport</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDeveloper</td>
<td>Oracle</td>
<td>September 2012, 11.1.2.3.0 Build 6276.1</td>
<td>Multiple</td>
<td>Proprietary OTN JDeveloper License (Freeware)</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, EP</td>
</tr>
<tr>
<td>Geany</td>
<td>Colomban Wendling Nick Treleaven Matthew Brush Enrico Tröger Frank Lanitz</td>
<td>June 2012, 1.22</td>
<td>Multiple</td>
<td>GPL</td>
<td>Yes, EP</td>
<td>No</td>
<td>Yes, EP</td>
</tr>
<tr>
<td>IntelliJ IDEA</td>
<td>JetBrains</td>
<td>July 2012, 11.1.3 Build: 117.798</td>
<td>Multiple</td>
<td>ALv2, proprietary</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Aptana</td>
<td>Aptana, Inc.</td>
<td>August 2012, 3.2.2</td>
<td>Multiple</td>
<td>GPL, proprietary</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes, EP</td>
</tr>
<tr>
<td>NetBeans</td>
<td>Oracle</td>
<td>August 2012, 7.2</td>
<td>Multiple</td>
<td>CDDL, GPL2</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Eclipse Foundation</td>
<td>4.2.1</td>
<td>Multiple</td>
<td>GPL</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Table: IDE Comparison Overview (I)

The available environments are then compared with respect to documentation and community support, functions/plugins relevant to the Generic Service Development Platform and their collaborative characteristics.

<table>
<thead>
<tr>
<th>IDE</th>
<th>Documentation and community support</th>
<th>Functionalities relevant to Development Platform</th>
<th>Collaboration</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDeveloper</td>
<td>Extensive</td>
<td>Limited</td>
<td>Oracle Team Productivity Software</td>
</tr>
<tr>
<td>Geany</td>
<td>Small</td>
<td>Very limited</td>
<td>Not supported</td>
</tr>
<tr>
<td>IntelliJ IDEA</td>
<td>Moderate</td>
<td>Limited</td>
<td>Jethbrains-developed Team City, Youtrack</td>
</tr>
<tr>
<td>Aptana</td>
<td>Moderate</td>
<td>No</td>
<td>Git integration only</td>
</tr>
<tr>
<td>NetBeans</td>
<td>Moderate</td>
<td>No</td>
<td>Native Support</td>
</tr>
<tr>
<td>Eclipse</td>
<td>Extensive</td>
<td>Yes, MDA, Automated Code Generation, Models and tools to model, simulate, test and execute business artefacts</td>
<td>Several plugins, providing versioning control, workspaces, etc</td>
</tr>
</tbody>
</table>

Table: IDE Comparison overview (II)
We have attempted this comparison in order to select an appropriate IDE which supports a majority of the functional and non-functional requirements of the Generic Service Development platform.

From the analysis performed above, the Eclipse IDE seems to be the best candidate for the technology foundation of the Generic Service Development platform. It combines the following set of characteristics that are necessary for the foundation of the Generic Service Development platform:

- Open source.
- Visual Programming Environment
- Open plugin architecture allows extensive customisations with minimal integration overheads.
- Very extensive range of plugins, supporting automatic code generation, Model Driven Architecture and Development, Transformations, Collaboration, GUI creation, etc. The plugins can be mapped to almost all required functionalities.
- Extensive documentation and support from official sources, organisations, as well as the user community.