

A PROCESS MODELING BASED METHOD FOR IDENTIFICATION AND  
IMPLEMENTATION OF SOFTWARE DEVELOPMENT TOOL INTEGRATION-  
TUPLES

A THESIS SUBMITTED TO  
THE GRADUATE SCHOOL OF INFORMATICS  
OF  
THE MIDDLE EAST TECHNICAL UNIVERSITY

BY

K. ALPAY ERTÜRKMEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF  
DOCTOR OF PHILOSOPHY  
IN  
THE DEPARTMENT OF INFORMATION SYSTEMS

MARCH 2010

Approval of the Graduate School of Informatics

\_\_\_\_\_  
Prof. Dr. Nazife BAYKAL

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Doctor of Philosophy.

\_\_\_\_\_  
Assist. Prof. Dr. Tuğba TAŞKAYA TEMİZEL

Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Doctor of Philosophy.

\_\_\_\_\_  
Assoc. Prof. Dr. Onur DEMİRÖRS

Supervisor

Examining Committee Members

Prof. Dr. Semih BİLGEN (METU, EEE) \_\_\_\_\_

Assoc. Prof. Dr. Onur DEMİRÖRS (METU, II) \_\_\_\_\_

Dr. Kıvanç Dinçer (BILKENT, CS) \_\_\_\_\_

Assist. Prof. Dr. Kayhan İmre (HACETTEPE, BİL) \_\_\_\_\_

Assist. Prof. Dr. Altan Koçyiğit (METU, II) \_\_\_\_\_

**I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.**

**Name, Last name: K. ALPAY ERTÜRKMEN**

**Signature :**

## **ABSTRACT**

### **A PROCESS MODELING BASED METHOD FOR IDENTIFICATION AND IMPLEMENTATION OF SOFTWARE DEVELOPMENT TOOL INTEGRATION- TUPLES**

Ertürkmen, K. Alpay

Ph.D., Department of Information Systems

Supervisor: Assoc. Prof. Dr. Onur Demirörs

March 2010, 223 pages

Software development is highly dependent on the use of tools. These tools support and automate activities performed in different sub-domains of software development. However, they don't adequately provide or support integration facilities, and act as "islands of automation". This restricts their benefits to only specific parts of the process. To reap the benefits of integration, this thesis provides a process modeling based method named PLETIN to identify and implement software development tool integration-tuples. The method aims to present solutions for issues observed in tool integration for software development organizations by delivering an integrated tool set. The proposed solution approach is based on the idea that if there were no integrations between tools at all, users would perform the necessary actions to cooperate different tools. PLETIN is a method for the identification of the candidate integration situations (integration-tuples) from the interactions of users with the tools. These tuples constitute the requirements used to develop integration facilities.

The software development process definitions are used as inputs to create process models and provide actual implementations. The research is supported with case-study work to identify the significance of the problems and the applicability of the method as a solution to issues in tool integration.

Keywords: software development process, software development tools, tool integration, process modeling

## ÖZ

### YAZILIM GELİŞTİRME ARAÇ ENTEGRASYONLARININ AYIRT EDİLMESİ VE UYGULANMASI İÇİN SÜREÇ MODELLEME TABANLI BİR METOD

Ertürkmen, K. Alpay

Doktora, Bilişim Sistemleri Bölümü

Tez Yöneticisi: Doç. Dr. Onur Demirörs

Mart 2010, 223 sayfa

Yazılım geliştirme süreci çeşitli araçların kullanımına ciddi anlamda bağımlılık gösterir. Bu araçlar yazılım geliştirme sürecinin farklı alt-alanlarında gerçekleştirilen işleri destekler ve otomatikleştirir. Fakat bu araçlar yeterli entegrasyon imkanlarını sağlamayarak ya da desteklemeyerek birer “otomasyon adası” olarak davranmaktadır. Bu davranış, araçların faydalarının sürecin sadece belirli parçalarına kısıtlanmasına sebep olmaktadır. Entegrasyonun faydalarından yararlanılabilmesi için bu tez, yazılım geliştirme araç entegrasyonlarının tanımlanması ve gerçekleştirilmesi için kullanılan süreç modelleme tabanlı PLETIN adında bir metod sunmaktadır. Bahsi geçen metod, entegre bir araç grubu oluşturulmasını yardımcı olarak yazılım geliştiren kurumların yaşadığı araç entegrasyonu temelli sorunlara çözümler sağlamayı hedeflemektedir. Önerilen çözüm yaklaşımı, eğer ortamda herhangi bir entegrasyon olmasaydı farklı araçları birarada çalıştırabilmek için gereken işlemleri kullanıcıların yapması gerektiği varsayımına dayanmaktadır. PLETIN kullanıcıların araçlar ile etkileşimlerinden, aday entegrasyon durumlarının

tanımlanmasını mümkün kılmaktadır. Bu durumlar entegrasyonların gerçekleştirilmesinde temel alınan gereksinimleri oluşturur. Yazılım geliştirme süreç tanımları girdi olarak kullanılıp süreç modelleri ve gerçek uygulamalar geliştirilir. Yapılan araştırma, sorunların ciddiyeti ve metodun araç entegrasyonu problemine uygulanabilirliğinin anlaşılabilmesi için durum-çalışması ile desteklenmiştir.

Anahtar Kelimeler: yazılım geliştirme süreci, yazılım geliştirme araçları, araç entegrasyonu, süreç modelleme.

To my beloved wife, Gökçe Banu



## **ACKNOWLEDGEMENTS**

First of all, I would like to express my gratitude to my supervisor Assoc. Prof. Dr. Onur Demirörs for his guidance throughout my graduate studies.

I wish to express a lot of thanks to Prof. Dr. Semih Bilgen and Assist. Prof. Dr. Altan Koçyiğit for their valuable suggestions and comments throughout the steering meetings of this study.

I would also like to thank Dr. Kıvanç Dinçer and his team, Öcal Fidanboy, Burcu Akkan, Cenkler Yakın for their support on the case studies.

I also want to express my gratefulness to my wife Gökçe Banu for all her patience, love, support and tolerance. Without her understanding this work would never been possible.

Finally, I would like to thank to my parents and dear sister, for their support and love.

## TABLE OF CONTENTS

<b>ABSTRACT .....</b>	<b>iv</b>
<b>ÖZ.....</b>	<b>vi</b>
<b>ACKNOWLEDGEMENTS .....</b>	<b>ix</b>
<b>TABLE OF CONTENTS .....</b>	<b>x</b>
<b>LIST OF TABLES .....</b>	<b>xiii</b>
<b>LIST OF FIGURES .....</b>	<b>xiv</b>
<b>CHAPTER</b>	
<b>1. INTRODUCTION.....</b>	<b>1</b>
<b>1.1. The Context.....</b>	<b>7</b>
<b>1.2. Solution Approach.....</b>	<b>8</b>
<b>1.3. Contributions .....</b>	<b>10</b>
<b>1.4. Organization of the Thesis.....</b>	<b>12</b>
<b>2. RELATED RESEARCH .....</b>	<b>14</b>
<b>2.1. Tool Integration Approaches.....</b>	<b>15</b>
2.1.1. Early Message-Passing/Control Integration Approaches.....	18
2.1.2. Application Lifecycle Framework (ALF) .....	20
2.1.3. ToolNet System.....	22
2.1.4. Data-Sharing Approaches .....	24
2.1.5. Agent-based Approaches .....	25
2.1.6. Process Centered Software Engineering Environments (PCSEEs).....	27
<b>3. THE PLETIN METHOD .....</b>	<b>29</b>
<b>3.1. Modeling Approach.....</b>	<b>33</b>
<b>3.2. Method Stages.....</b>	<b>36</b>

<b>3.3.</b>	<b>Context Definition (Stage I)</b> .....	<b>37</b>
<b>3.4.</b>	<b>Process Definition (Stage II)</b> .....	<b>39</b>
3.4.1.	Actor and Action Identification .....	42
3.4.2.	Process Flow Identification.....	44
3.4.3.	Tool and Tool Interaction Identification .....	45
<b>3.5.</b>	<b>Process Mapping (Stage III)</b> .....	<b>47</b>
3.5.1.	Identification of Atomic Actions .....	47
3.5.2.	Identification of Implicit Sequences .....	49
3.5.3.	Development of Custom Integration Implementations .....	54
<b>3.6.</b>	<b>Process Execution (Stage IV)</b> .....	<b>56</b>
<b>3.7.</b>	<b>Roles</b> .....	<b>59</b>
<b>3.8.</b>	<b>Notation</b> .....	<b>60</b>
<b>3.9.</b>	<b>Comparison of the PLETIN Method with Previous Efforts</b> .....	<b>60</b>
<b>3.10.</b>	<b>Limitations of PLETIN</b> .....	<b>63</b>
<b>4.</b>	<b>ENABLING TECHNOLOGIES</b> .....	<b>66</b>
<b>4.1.</b>	<b>Business Process Modeling Notation (BPMN)</b> .....	<b>66</b>
4.1.1.	Choice of BPMN.....	67
4.1.2.	BPMN Elements .....	68
<b>4.2.</b>	<b>Intalio BPM Community Edition</b> .....	<b>74</b>
<b>4.3.</b>	<b>Eclipse and Apache Tomcat</b> .....	<b>78</b>
<b>5.</b>	<b>CASE STUDIES</b> .....	<b>79</b>
<b>5.1.</b>	<b>Multiple Case Study Design</b> .....	<b>81</b>
5.1.1.	Case Study I Design .....	81
5.1.2.	Case Study II Design.....	84
<b>5.2.</b>	<b>Case Selection</b> .....	<b>85</b>
<b>5.3.</b>	<b>Execution of Case Study I</b> .....	<b>85</b>
5.3.1.	Tool Use Exploration Phase.....	86
5.3.2.	Identification Phase.....	87
5.3.3.	Implementation Phase .....	90
5.3.4.	Discussion on Implicit Sequences for Case Study I.....	93
<b>5.4.</b>	<b>Results for Case Study I</b> .....	<b>95</b>
<b>5.5.</b>	<b>Execution of Case Study II</b> .....	<b>101</b>
5.5.1.	Context Definition Phase .....	103

5.5.2.	Process Definition Phase.....	106
5.5.3.	Process Mapping Phase.....	107
5.5.4.	Discussion on Implicit Sequences for Case Study II .....	108
<b>5.6.</b>	<b>Results for Case Study II .....</b>	<b>110</b>
<b>5.7.</b>	<b>Validity Threats.....</b>	<b>112</b>
<b>5.8.</b>	<b>Discussion.....</b>	<b>113</b>
<b>6.</b>	<b>CONCLUSIONS .....</b>	<b>118</b>
	<b>REFERENCES.....</b>	<b>123</b>
	<b>APPENDICES</b>	
	<b>APPENDIX A: COMPLETE LIST OF OPERATIONS DERIVED FROM PROCESS MODELS (CASE STUDY I) .....</b>	<b>131</b>
	<b>APPENDIX B: COMPLETE LIST OF OPERATIONS DERIVED FROM PROCESS MODELS (CASE STUDY II).....</b>	<b>133</b>
	<b>APPENDIX C: PROCESS LIST (CASE STUDY I).....</b>	<b>134</b>
	<b>APPENDIX D: PROCESS MAPPING (CASE STUDY II).....</b>	<b>137</b>
	<b>APPENDIX E: PROCESS MODELS (CASE STUDY I).....</b>	<b>141</b>
	<b>APPENDIX F: PROCESS MODELS (CASE STUDY II) .....</b>	<b>165</b>
	<b>APPENDIX G: DEFINITIONS FOR WEB SERVICES (CASE STUDY I)...</b>	<b>191</b>
	<b>APPENDIX H: APPLICATION CODE DEVELOPED FOR WEB SERVICES (CASE STUDY I) .....</b>	<b>213</b>
	<b>VITA.....</b>	<b>223</b>

## LIST OF TABLES

<i>Table 1 Types of process with respect to the number of tool interactions .....</i>	<i>37</i>
<i>Table 2 Sample process list for RE process area.....</i>	<i>38</i>
<i>Table 3 Types of implicit sequences.....</i>	<i>50</i>
<i>Table 4 BPMN elements.....</i>	<i>68</i>
<i>Table 5 Processes areas not directly related to software development.....</i>	<i>86</i>
<i>Table 6 Process areas included in the scope of Case Study I.....</i>	<i>87</i>
<i>Table 7 Tool interactions with respect to process areas.....</i>	<i>95</i>
<i>Table 8 Distribution of candidate integration sequences with respect to process areas .....</i>	<i>96</i>
<i>Table 9 Distribution of tuples with respect to process areas .....</i>	<i>96</i>
<i>Table 10 Execution frequency of tuple.....</i>	<i>97</i>
<i>Table 11 Distribution of operations to tools .....</i>	<i>100</i>
<i>Table 12 Differences between the two target organizations .....</i>	<i>102</i>
<i>Table 13 List of processes selected for analysis.....</i>	<i>105</i>

## LIST OF FIGURES

<i>Figure 1 Types of tools.....</i>	<i>2</i>
<i>Figure 2 A fictional monolithic tool and its process support.....</i>	<i>4</i>
<i>Figure 3 A fictional tool suite and its process support .....</i>	<i>5</i>
<i>Figure 4 A fictional tool set from several vendors with point-to-point integrations.....</i>	<i>6</i>
<i>Figure 5 Process model for PLETIN.....</i>	<i>11</i>
<i>Figure 6 Artifact relationships in PLETIN .....</i>	<i>12</i>
<i>Figure 7 Control integration/Data sharing approach [13] .....</i>	<i>19</i>
<i>Figure 8 ALF architecture [6].....</i>	<i>20</i>
<i>Figure 9 ALF mechanism [6].....</i>	<i>21</i>
<i>Figure 10 ToolNet architecture .....</i>	<i>22</i>
<i>Figure 11 Tool integration sophistication vs. sustainability [1].....</i>	<i>23</i>
<i>Figure 12 Toaster model [62].....</i>	<i>24</i>
<i>Figure 13 Data sharing approach .....</i>	<i>25</i>
<i>Figure 14 Architecture for agent-based tool integration.....</i>	<i>26</i>
<i>Figure 15 Process model for PLETIN.....</i>	<i>31</i>
<i>Figure 16 Conceptual map for the PLETIN method .....</i>	<i>35</i>
<i>Figure 17 Process model for PLETIN Stage I, Context Definition.....</i>	<i>40</i>
<i>Figure 18 Relationships of PLETIN BPMN Elements .....</i>	<i>41</i>
<i>Figure 19 Process model for PLETIN Stage II, Process Definition .....</i>	<i>43</i>
<i>Figure 20 Sample User Representations on Process Model.....</i>	<i>44</i>
<i>Figure 21 Sample process flow with BPMN notation .....</i>	<i>45</i>
<i>Figure 22 Sample tool interaction represented as a process model .....</i>	<i>46</i>
<i>Figure 23 Two actions grouped into a sequence .....</i>	<i>46</i>
<i>Figure 24 Process model for PLETIN Stage III, Process Mapping.....</i>	<i>48</i>
<i>Figure 25 Sequence breakdown.....</i>	<i>49</i>
<i>Figure 26 A sample interrupted implicit sequence .....</i>	<i>51</i>
<i>Figure 27 A sample compound implicit sequence.....</i>	<i>52</i>
<i>Figure 28 Placeholder tasks for tools.....</i>	<i>55</i>
<i>Figure 29 Web-service invocations by the process manager.....</i>	<i>58</i>

<i>Figure 30 Supported BPMN elements in Intalio Designer .....</i>	<i>74</i>
<i>Figure 31 Web-service definitions imported to the workspace .....</i>	<i>74</i>
<i>Figure 32 Data mapping in Intalio Designer .....</i>	<i>75</i>
<i>Figure 33 Intalio Designer process deployment dialog .....</i>	<i>75</i>
<i>Figure 34 Intalio BPM Community Edition process operations interface.....</i>	<i>76</i>
<i>Figure 35 Intalio BPM Community Edition process detail interface.....</i>	<i>77</i>
<i>Figure 36 Intalio BPM Community Edition process instance detail interface .....</i>	<i>77</i>
<i>Figure 37 Eclipse "web-service wizard" .....</i>	<i>78</i>
<i>Figure 38 Process model for Case Study I Design .....</i>	<i>82</i>
<i>Figure 39 Sample BPMN model for Case Study I, Phase I, RE5214.....</i>	<i>89</i>
<i>Figure 40 Sample tool interaction represented as a process model .....</i>	<i>90</i>
<i>Figure 41 Two actions grouped into a sequence .....</i>	<i>91</i>
<i>Figure 42 Web service definitions imported to the workspace .....</i>	<i>92</i>
<i>Figure 43 Sample completed BPMN model for Case Study I, RE5214.....</i>	<i>92</i>
<i>Figure 44 Process model for TS512.....</i>	<i>93</i>
<i>Figure 45 Process model for TS521.....</i>	<i>94</i>
<i>Figure 46 Integration map for the case study .....</i>	<i>98</i>
<i>Figure 47 Existing integration map of the organization.....</i>	<i>98</i>
<i>Figure 48 Number of tuples constituting the integration map .....</i>	<i>99</i>
<i>Figure 49 Process model of Case Study II.....</i>	<i>101</i>
<i>Figure 50 Process model of Case Study II, Phase I.....</i>	<i>104</i>
<i>Figure 51 Process model of Case Study II, Phase II.....</i>	<i>106</i>
<i>Figure 52 Sample process model for Case Study II, Phase II (UG-070-87).....</i>	<i>107</i>
<i>Figure 53 Sample process model for Case Study II, Phase III .....</i>	<i>108</i>
<i>Figure 54 Process model for KY-020-62142 .....</i>	<i>109</i>
<i>Figure 55 Sequence KY-020-62142 decomposed.....</i>	<i>110</i>
<i>Figure 56 Process model of Case Study II, Phase III .....</i>	<i>111</i>
<i>Figure 57 Effort distribution for Case Study II.....</i>	<i>112</i>
<i>Figure 58 Process list (CASE STUDY I) - Part 1 .....</i>	<i>135</i>
<i>Figure 59 Process list (CASE STUDY I) - Part 2 .....</i>	<i>136</i>
<i>Figure 60 Process Mapping for Case Study II – Part 1.....</i>	<i>138</i>
<i>Figure 61 Process Mapping for Case Study II - Part 2 .....</i>	<i>139</i>
<i>Figure 62 Process Mapping for Case Study II - Part 2 .....</i>	<i>140</i>
<i>Figure 63 Process Model for BRPG2 - Part 1.....</i>	<i>142</i>
<i>Figure 64 Process Model for BRPG2 - Part 2.....</i>	<i>143</i>
<i>Figure 65 Process Model for CMG510.....</i>	<i>144</i>
<i>Figure 66 Process Model for CM513 .....</i>	<i>145</i>

<i>Figure 67 Process Model for CM533 .....</i>	<i>146</i>
<i>Figure 68 Process Model for CMG21-Part 1 .....</i>	<i>147</i>
<i>Figure 69 Process Model for CG21-Part 2 .....</i>	<i>148</i>
<i>Figure 70 Process Model for CMG21-Part 3 .....</i>	<i>149</i>
<i>Figure 71 Process Model for CMTG211 .....</i>	<i>150</i>
<i>Figure 72 Process Model for CMTG212 .....</i>	<i>151</i>
<i>Figure 73 Process Model for CMTG213-216 Part 1 .....</i>	<i>152</i>
<i>Figure 74 Process Model for CMTG21-216 Part2 .....</i>	<i>153</i>
<i>Figure 75 Process Model for RE5213.....</i>	<i>154</i>
<i>Figure 76 Process Model for RE5214.....</i>	<i>155</i>
<i>Figure 77 Process Model for RE52212 Part 1.....</i>	<i>156</i>
<i>Figure 78 Process Model for RE52212 Part 2.....</i>	<i>157</i>
<i>Figure 79 Process Model for RE5222.....</i>	<i>158</i>
<i>Figure 80 Process Model for RMTG21.....</i>	<i>159</i>
<i>Figure 81 Process Model for RMTG22.....</i>	<i>160</i>
<i>Figure 82 Process Model for RMTG23.....</i>	<i>161</i>
<i>Figure 83 Process model for TS514.....</i>	<i>162</i>
<i>Figure 84 Process Model for TS524 .....</i>	<i>163</i>
<i>Figure 85 Process Model for VV542 .....</i>	<i>164</i>
<i>Figure 86 Process Model for KY-020-621-Part 1.....</i>	<i>166</i>
<i>Figure 87 Process Model for KY-020-621-Part 2.....</i>	<i>167</i>
<i>Figure 88 Process Model for KY-020-621-Part 3.....</i>	<i>168</i>
<i>Figure 89 Process Model for KY-020-621-Part 4.....</i>	<i>169</i>
<i>Figure 90 Process Model for UG-010-84 .....</i>	<i>170</i>
<i>Figure 91 Process Model for UG-040-83 .....</i>	<i>171</i>
<i>Figure 92 Process Model for UG-070-81 .....</i>	<i>172</i>
<i>Figure 93 Process Model for UG-070-82.....</i>	<i>173</i>
<i>Figure 94 Process Model for UG-070-83.....</i>	<i>174</i>
<i>Figure 95 Process Model for UG-070-86.....</i>	<i>175</i>
<i>Figure 96 Process Model for UG-070-87.....</i>	<i>176</i>
<i>Figure 97 Proces Model for UG-070-89.....</i>	<i>177</i>
<i>Figure 98 Process Model for UG-190-82 .....</i>	<i>178</i>
<i>Figure 99 Process Model for UG-190-89-Part1.....</i>	<i>179</i>
<i>Figure 100 Process Model for UG-190-89-Part2.....</i>	<i>180</i>
<i>Figure 101 Process Model for UG-190-89-Part3.....</i>	<i>181</i>
<i>Figure 102 Process Model for UG-190-810-Part 1.....</i>	<i>182</i>
<i>Figure 103 Process Model for UG-190-810-Part 2.....</i>	<i>183</i>



<i>Figure 104 Process Model for UG-190-810-Part 3.....</i>	<i>184</i>
<i>Figure 105 Process Model for UG-190-810-Part 4.....</i>	<i>185</i>
<i>Figure 106 Process Model for UG-190-811.....</i>	<i>186</i>
<i>Figure 107 Process Model for UG-190-812-Part 1.....</i>	<i>187</i>
<i>Figure 108 Process Model for UG-190-812-Part 2.....</i>	<i>188</i>
<i>Figure 109 Process Model for UG-190-813-Part 1.....</i>	<i>189</i>
<i>Figure 110 Process Model for UG-190-813-Part 2.....</i>	<i>190</i>

# **CHAPTER 1**

## **INTRODUCTION**

The scope of software development in modern organizations is getting broader as the business needs and software complexity increases. Once formally defined only as design and coding of software systems, software development now encompasses planning, requirements definition, requirements management, design, coding, building, testing, configuration management and maintenance of software systems.

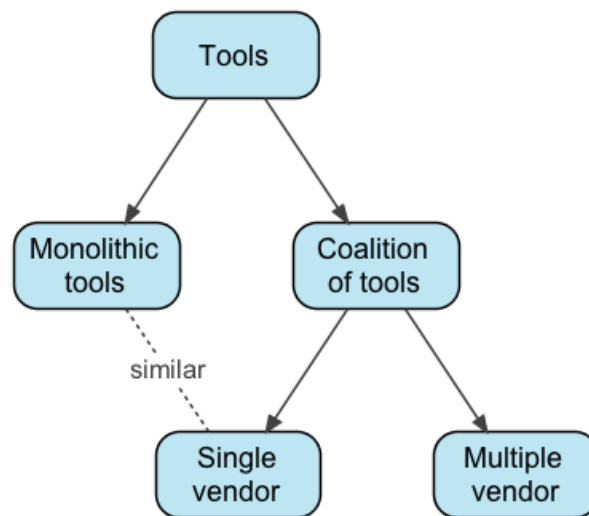
As a direct consequence of the widening in the scope of software development processes, the number of stakeholders, the complexity of the development processes and the effort spent increases.

Software development has thus become a complex sequence of information transformations, with a pre-defined aim and several levels of input and knowledge [9]. These information transformations create outputs that are used as inputs in succeeding steps towards the goal. These transformations are specialized into separate processes like requirements engineering, software design, coding, and software testing. As in any engineering domain, tools have been developed to support software engineers by increasing the efficiency of the execution of processes.

Tools are used to handle the complexity surrounding software development processes [38]. There is much evidence in the literature on how tool use provides benefits for software development in terms of quality and cost [36], [57], [45].

Tools that support software development can be classified into two groups (See Figure 1): the first group contains monolithic development benches created and supported by a single vendor or organization [34]. These development benches target a single platform, and are designed to support as much of the whole software development process as possible. Monolithic tools generally only support specific development technologies and target platforms. They are large, very complex and do not provide flexibility. They are costly to build and acquire, hard to maintain and modify for different goals.

The second group is composed of individual tools supporting one or more discrete software development phases/sub-domains [60]. Software development organizations targeting different platforms/technologies or those operating in heterogeneous environments (like complex enterprise applications or open systems) require a variety of tools. Monolithic tools do not provide support for a mix of target platforms/technologies and are not suitable for modification.



**Figure 1 Types of tools**

Vendors develop distinct tools for different platforms and technologies that can be used to operate in heterogeneous environments. Another source for these tools are communities of open source developers. These tools range from simple time tracking solutions to complex continuous integration systems. They are specialized to support

and automate specific (or several) sub-processes of software development, like requirements management or version control [45]. Being specialized on supporting and automating certain parts of the complete process, they constitute “islands of automation” if they do not provide sufficient integration facilities [63], [54], [36].

[45] and [38] state that, "for improved productivity, quality and reduced risk, IT infrastructures need to be highly integrated and interoperable". [63] defines tool integration as:

“the techniques used to form coalitions of tools to provide an environment that supports some, or all, of the activities within a software engineering process”.

Wasserman [60] identifies tool integration as:

“an intention to produce complete environments that support the entire software development lifecycle”.

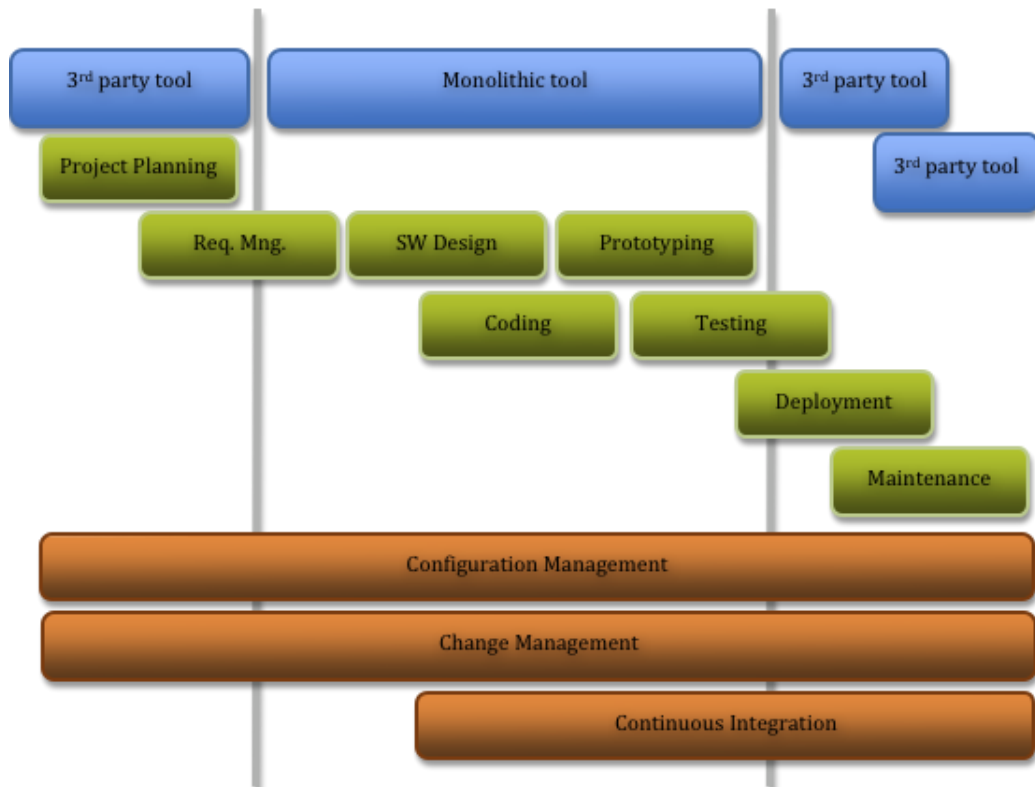
Thomas and Nejme [57] developed a more specific approach stating:

“Tool integration is about the extent to which tools agree. The subject of these agreements may include data format, user-interface conventions, use of common functions, or other aspects of tool construction”.

Monolithic tools are developed by a single group, and are inherently integrated. They aim to support the whole software development process, but generally support a fraction of it in practice. They also have the following limitations:

1. Monolithic tools are expensive to develop and acquire, because they are large and complex.
2. Monolithic tools don't support interchangeable components by definition. They have rigid structures that are not interoperable or interchangeable with 3<sup>rd</sup> party components. Organizations using monolithic tools become “vendor-dependant” because of this limitation.
3. Monolithic tools aim to support the complete development process, but end up supporting a fraction of it (See Figure 2).

4. Monolithic tools do not support different technologies and platforms.



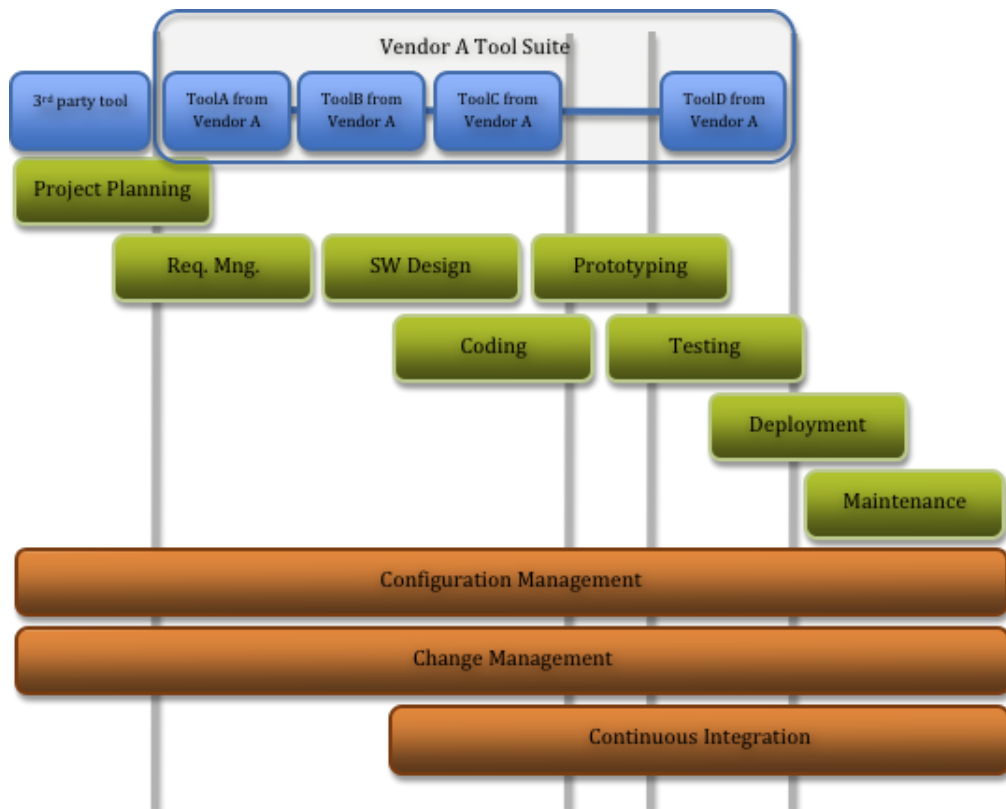
**Figure 2 A fictional monolithic tool and its process support**

These constraints render monolithic tools an infeasible solution for most organizations. The alternative for these organizations is to use separate tools to support different sub-domains, technologies, and platforms.

Integration of discrete tools that support different sub-domains is not trivial and presents its own challenges. Integration between the tools is mostly realized by tool-vendors, or in some rare cases by independent 3<sup>rd</sup> party developers.

Integration implementations developed by vendors are strategic and favor the vendor's own set of tools. They are used to establish a suite of integrated tools [14]. An integrated tool suite resembles a monolithic tool, which is developed from the ground-up by a single vendor to support the whole process (See Figure 3). However since each organization has different requirements stemming from varying organizational processes, customers, technologies, and target platforms, tool suites can be rarely satisfactory. A rigid, one-size-fits-all solution is not acceptable for the

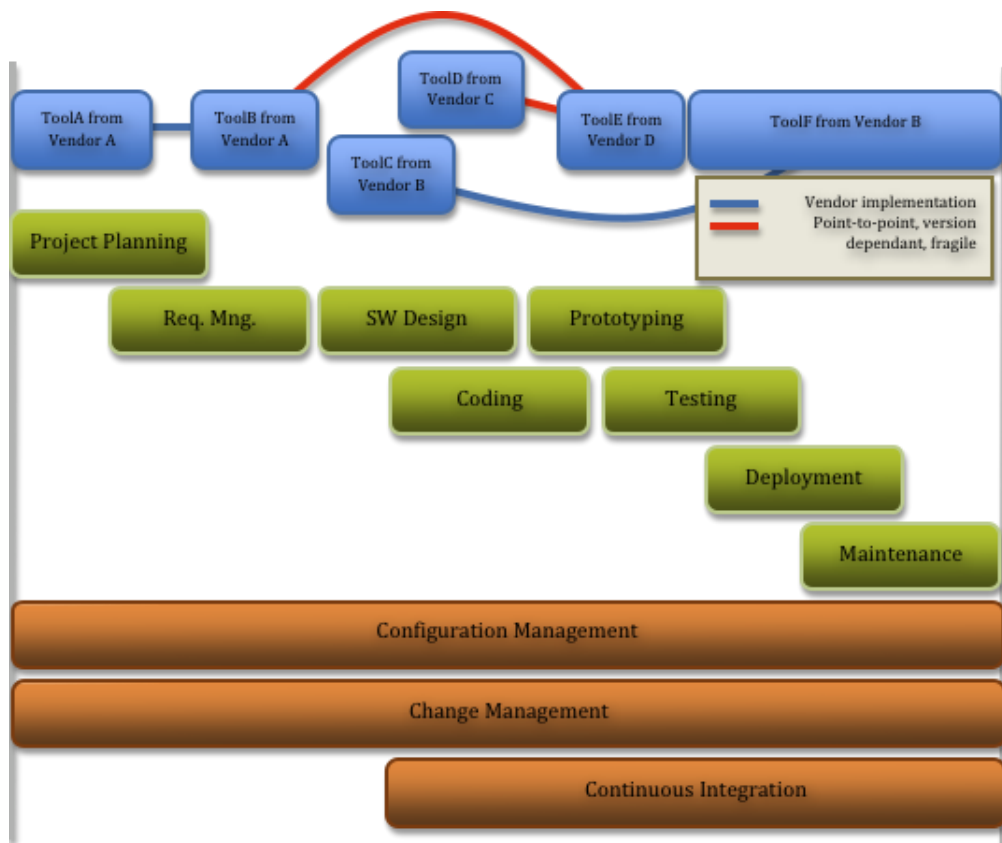
variety of requirements. These solutions in the form of monolithic tools and tool suites prevent organizations from creating their own tool sets based on their own organizational requirements and constraints [1]. Organizations can't use tools they choose but have to depend on bundles designed and provided by vendors. This is called vendor-dependency. The choice of a single vendor tool suite can even constrain the platforms and technologies an organization can support and operate in.



**Figure 3 A fictional tool suite and its process support**

Occasionally, vendors develop integration implementations to support tools from other vendors. These mostly originate from strategic relationships between vendors. Being integrated to a very popular commercial tool or widely used open source software can positively influence the market for the tool, and this can be the reason for intra-vendor integration implementations. Depending on this kind of integrations between the tools, organizations can develop tool sets satisfying their requirements (See Figure 4).

These implementations are generally specific to a particular version of the tools. Since there is no standard framework of defined and widely accepted interfaces, tools do not provide standardized interfaces. Integration implementations built to support a specific version of a tool can become obsolete as their internals change with new iterations. Organizations depending on these integration facilities are locked down to certain older versions and they can't upgrade their tool infrastructure if they can't give up the functionality provided by the integration [54]. This is called "version-dependency". These point-to-point (bilateral) integration implementations are "fragile, partial and inflexible" [38], [1], [13]. This limits the organizations' freedom of choosing best in class, most suitable and economic technologies and tools to develop their own tool set. The resulting software development infrastructure is rigid, inferior, expensive, hard to maintain and vendor-dependant, while the number of implementations required increase exponentially with the number of tools in use.



**Figure 4 A fictional tool set from several vendors with point-to-point integrations**

## 1.1. The Context

The literature survey performed for this thesis unveiled many approaches to tool integration in literature. [62] provides an excellent bibliography of research with significance in the domain. Research on tool integration is mostly formulative, focused mostly on providing solutions, rather than descriptive, focused on understanding and describing the domain [63]. This makes it harder to categorize proposed solutions, compare and evaluate them and develop new solutions based on existing ones. However, there are several proposed categorizations for tool interaction including [63] and [44], which are detailed in Chapter 2. Broadly, tool integration efforts range from standardization efforts and architectural models to modern XML/XMI oriented approaches. These efforts define guidelines or standards on how should tools be built, how should they their structure be, and how should they communicate.

Wasserman, in his 1989 paper [60] defines five types of tool integration: platform, presentation, data, control and process. Most approaches in the literature focus on data integration, on how data is shared and objects are managed.

Previous research frequently emphasizes the importance of aligning tool integration with processes. However, an answer to the question “how tool integration can be developed based on organizational processes?” is often discarded, or solutions similar to Process Centered Software Engineering Environments (PCSEEs) that support processes without focus on integrating existing ones are proposed.

The work in this thesis was inspired by an Eclipse project named Application Lifecycle Framework (ALF) [6]. It is one of the most recent efforts on tool integration. Unfortunately it has been terminated before being finalized. The details of ALF are available in Section 2.1.2. The approach ALF takes is "to create a multi-layered interoperability framework leveraging SOA technologies". It is based on the orchestration of tools to provide processes that can be repetitively and efficiently executed. In short, the aim of the ALF project is to develop a standard-based tool integration environment. In this environment tools communicate using a common



vocabulary. The tools are expected to provide/expose ALF-compliant services so that they can be orchestrated to execute the processes.

Unfortunately, the ALF project was terminated while it is in Eclipse incubation stage, due to insufficient community participation except from tool vendor Serena [25]. The most important phase of the project, as stated in the “Termination Review”, was the determination of “a set of domain vocabularies that define the events, objects and attributes”. The vocabularies constitute a core component of services that are used to orchestrate the processes. This phase required a high-level of participation since the aim was to develop a common vocabulary that is widely accepted in the industry. The technique employed was to bring together experts from the industry, receive their opinions to start discussions and reach an agreement at later stages. However, the lack of participation from the community, and possible bias from the contribution of a single vendor resulted in the project being archived and the termination of further developments.

## **1.2. Solution Approach**

To prevent vendor and version dependency, a tool integration infrastructure that is based on open standards and open technologies is required. The integration infrastructure must empower organizations to develop a tool set satisfying their own requirements. Organizations must be able to choose best-in-class tools, and complement them with any other tool to develop tool sets that have sufficient features for them conforming to their economical constraints. The framework must support tool interoperability and interchangeability. Organizations must be able to change any tool they use with another one without much effort, and incorporate any tool to their tool set [1]. Today’s fast changing businesses mandates software to be flexible, adaptable and integrated. Tools infrastructures must be built to assist with the adaptation and integration process [36].

Software development cannot be imagined without the use of tools. Many software development organizations are already invested on tools to support their processes, integrated or not. Guidelines or models for better-integrated tool sets would not

provide benefits for organizations that already own tools. Solutions with a focus on existing tool sets would have a practical value.

Organizational processes affect the quality, cost and effort spent for software development. Many organizations are aware of the need to integrate the organization's process with tool support. Organizations even consider processes and tools to be inseparable [54]. The proposed solution must consider, be aligned to or even be based on organizational processes and tools must be integrated with respect to processes, rather than features of each other. This is stated by [14] as follows:

“tools are not simply integrated with each other, but are integrated with respect to specific process requirements. Further, entire tools are not integrated, but rather specific tool services (in the example, data flow diagram editing with documentation tool data interchange formats and document templates) are combined with some specific process result (production of standard documentation) to produce an integration of tool services.

While this n-ary relationship between tool services and process elements is conceptually tidy, in practice it is not easy to disentangle the process elements from the tool services (again, not surprising since CASE services tend to support end-user activities).”

There are two important questions left without emphasis in the literature:

1. How an already existing, much divergent tool set can be integrated?
2. What do the organizational processes expect from the tool set in terms of integration?

A solution answering these questions will have practical value in terms of being applicable to existing tool sets, while providing integration facilities that satisfy organizational requirements rather than fictional technical possibilities.

Briefly, a solution providing the following features is required:

1. Support open standards and technologies

2. Support tool interoperability and tool exchange (letting organizations choose whatever tools they see fit)
3. Support existing tools
4. Support organizational processes
5. Provide information for future tool developments

### **1.3. Contributions**

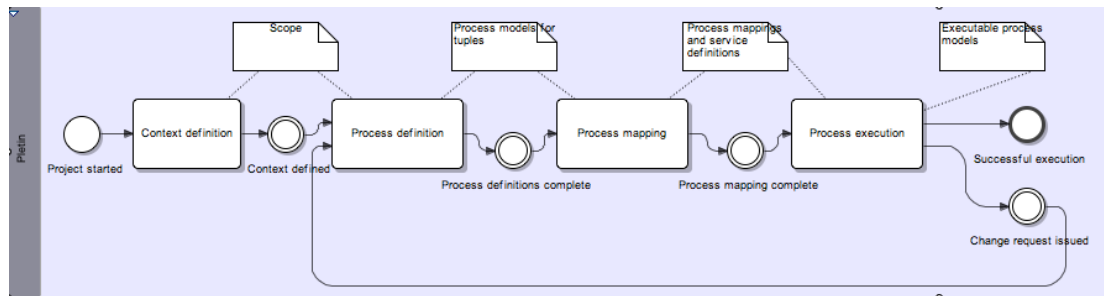
In this thesis, a method to derive the tool integration requirements of an organization from its software development processes is proposed. The proposed solution is based on the following perspective:

Assuming a situation where there are no integrations between the tools in a software development environment, cooperation of them must be maintained manually. As an example, to make it possible for different tools that are not integrated to work on the same data set, the data must be fed to each tool manually. Similarly, for a tool to operate on the information created by another tool, data should be moved between the tools by a user manually. In other words, users must perform actions necessary to keep the tools working together (cooperate). In this situation, process definitions (or models derived from these definitions if they exist) would contain sequences of actions (what we name integration-tuples or sequences, and use interchangeably in this thesis) performed to maintain tool integrations.

A process model is an abstract representation of software production activities and their relationship [9]. The investigation of these models can result in an understanding of how users interact with tools in software development to maintain non-existing tool integrations.

In the proposed method, process models are developed to visualize the process definitions. The integration requirements extracted from the process models are used to define and build custom interfaces for the tool set employed by the organization. Business processes are developed from process models, which mimic the manual

actions performed by users. These business processes consume the interfaces (implemented as web-services) developed for the tools when executed automatically. Thus, user actions are performed by the integration infrastructure on behalf of them and tools are integrated based on the requirements derived from organizational processes. See Figure 6 depicting the relationships of various artifacts used by the method.



**Figure 5 Process model for PLETIN**

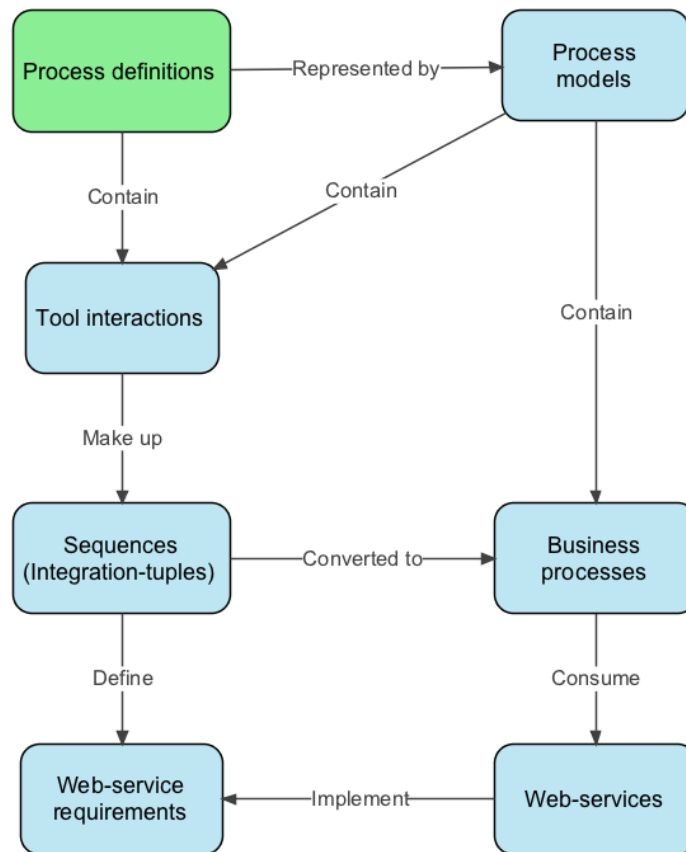
Ultimately, the integrations are realized as business processes that are executed automatically instead of manually by users. They have the following features:

1. They are developed based on organizational software development processes and process models representing them.
2. They executed automatically on a business process execution server.
3. They consume custom tool interfaces. These interfaces are developed for the tools so that they can satisfy integration requirements.

The conduct of the PLETIN method is detailed in Chapter 3. The process model for PLETIN is given in Figure 5. In the end, the users would observe that manual actions they carried out to maintain tool cooperation are no longer necessary. Rather, they can execute actions to affect multiple tools automatically. Individual tools would act as a coalition of integrated tools with the help of the automated business processes. Users would perceive the tool set as integrated.

Since tool integration in literature is mostly formulative, multiple case studies were performed to understand tool use in an organization, and the extent of the problems with tool integration. Based on this knowledge, the proposed method named PLETIN

(Process Level Tool Integration) was developed. The method was applied at two different organizations to identify integration requirements and develop a prototype implementation. Feedback obtained was used to improve the method and evaluate its feasibility.



**Figure 6 Artifact relationships in PLETIN**

## 1.4. Organization of the Thesis

The remainder of the thesis is organized into five chapters.

In Chapter 2, related research on software development tool integration approaches are described. Different approaches are compared to our approach to identify their advantages and limitations.

Chapter 3 describes the PLETIN method proposed by this thesis in detail. Each stage of the method, activities performed in these stages, inputs and outputs are discussed.

Chapter 4 discusses the technologies enabling the implementation of the PLETIN method.

Chapter 5 presents the multiple case study approach taken to understand the domain, to develop the method and evaluate it. Design and execution details of the case studies are given along with results and discussion.

Chapter 6 presents the conclusions reached and summarizes the contribution of this research. New questions that are raised by our research and the subjects that require further investigation are also described in this chapter.

## **CHAPTER 2**

### **RELATED RESEARCH**

Wicks and Dewar provide a detailed background on tool integration in literature [62]. They define tool integration in software engineering as [63]:

“the techniques used to form coalitions of tools to provide an environment that supports some, or all, of the activities within a software engineering process”

Wasserman identifies “the desire to link tools” in a software engineering environment as a “key issue” and defines it as supporting the entire software development lifecycle. Integration efforts aim to bring together tools supporting and benefitting the complete lifecycle of software development “through automation, with consequent productivity and quality improvements [63].

Thomas and Nejme [57] developed a more specific approach stating:

“Tool integration is about the extent to which tools agree. The subject of these agreements may include data format, user-interface conventions, use of common functions, or other aspects of tool construction”.

## 2.1. Tool Integration Approaches

Brown [13] states tool integration can be defined in two levels: conceptual (what is tool integration?) and mechanical (how do we provide integration?). He also categorizes tool integration literature into three groups:

1. New mechanisms and formulations for tool integration.
2. Examining semantics of tool integration including works of Wasserman [60], Thomas and Nejme [57], Wallnau and Feiler [14], [15].
3. Analysis of the relationship between integration and process (where little work is available).

Wasserman suggests a conceptual categorization in his paper [60] with the following types of tool integration:

1. Platform integration: various tools should be interoperable.
2. Presentation integration: tools should share a common “look and feel”.
3. Data integration: tool integration requires both sharing of data among tools and managing the relationships among data objects produced by different tools.
4. Control integration: tools should also be able to notify one another of events.
5. Process integration: major benefits from tools are achieved when they are used to support a well-defined software engineering process.

Another conceptual categorization for tool integration is given in [13] where Brown defines five levels:

1. Carrier level: Tools have a common form for data exchange like a byte stream.
2. Lexical level: Tools have common lexical conventions, a vocabulary with no relationship between words.
3. Syntactic level: Tools have common schemas, common rules for the creation of data structures.



4. Semantic level: Tools have a common understanding of the shared data.
5. Method level: Tools have information on the environment and process they support.

According to Brown [13], two most observed mechanical approaches to tool integration are:

1. Data sharing, mostly “through a common database in which all tools deposit their data”, or through techniques like a common object models, interface languages, or message exchange formats.
2. Control integration, based on actions and control signals where software development is seen as “a collection of services provided by different tools”.

“Data integration” or “data sharing” is the most frequently used approach in the literature. However Brown suggests that “control integration” strategy based on message passing would be more effective [13]. In their discussion on the state-of-the-art of CASE technologies [19] suggests that tool integration must be placed within a context of an organizational framework.

Rader et al. [54] defines five different levels (situations as it is called in the paper) in which an organization may have a tool infrastructure:

1. Isolated CASE tools
2. Clusters of CASE tools
3. Migration toward framework-based integration technology (database or message-passing framework)
4. Loosely integrated collections of CASE tool clusters
5. Complete integrated CASE environment

Although most organizations Rader et al. observed aim for Situation 4, Situation 5 is the focus of research.

The mechanical categorization in the literature is very diverse because of very different approaches taken to provide tool integration. [63] categorizes formulative work for the tool integration problem into three separate groups:

1. Process Centered Software Engineering Environments including but not limited to MARVEL [43], SPADE-1 [8], ADELE [10] and agent-based approaches [68]. PCSEEs (See Section 2.1.6) take a process-oriented approach to software development based on Osterweil's work [52] and aim to develop an environment supporting software development by defining and imposing certain rules or guidance on processes. Although PCSEEs do not provide facilities for tool integration [33], they provide an integrated support environment for the processes using tools.
2. Contemporary XML/XMI (extensible Markup Language/XML Metadata Interchange) [67], [66] oriented approaches based on different XML or XML based interchange languages. These languages are used for data sharing, or meta-model exchange. They originate from CDIF (CASE Data Interchange Format) [55].
3. Novel approaches including the use of ontologies, web services, Internet-based services, agent-based architectures (See Section 2.1.5), viewpoints and the ECMA (European Computer Manufacturers Association) Toaster model [28].

[6] suggests another categorization of tool integration efforts:

1. Standardization efforts or middleware services (CAIS [49], PCTE [3], CDIF [55], CORBA [50], RTP OTIF [51], agent based [22], etc.) which provide a common data and control interface for different tools to operate.
2. Architecture models, infrastructures and tool suites (ECMA Toaster Model [23], ToolBus architecture [11] etc.) define how tools should be developed so that they can provide services necessary for tool integration.

3. Basic tool integration mechanism schemes (data sharing, data linkage, data interchange, message passing, publish/subscribe services [35], [56]) to facilitate tool integration.

Basically, all these categorizations can be reduced to fundamental integration mechanisms Brown suggested: “data sharing” and “control integration”. Integration schemes either provide a standardized way for tools to exchange information, or tools are viewed as services and actions in the environment trigger the use/invocation of them, respectively. Data sharing approaches result in a consistent and re-usable representation of information during software development, however they impose performance overhead on tools and the process of integration because of the “necessary agreement required between the tools to define a common syntax and semantics for their data (e.g. a common data schema)” [13]. Since the schema must be defined beforehand, it is harder to succeed with tools organizations already own. In control integration based approaches, tools communicate with each other directly by passing messages rather than using a shared data repository.

Although many integration frameworks have been proposed in the literature, none of them have been widely adopted in practice [63]. The industry is still relying on individual tools for specific sub-processes of software development. Their integration is performed in an inefficient point-to-point manner, resulting in vendor and version-dependency.

The next sections of this chapter detail approaches similar to or significant for the method proposed in this thesis while discussing advantages and limitations of each.

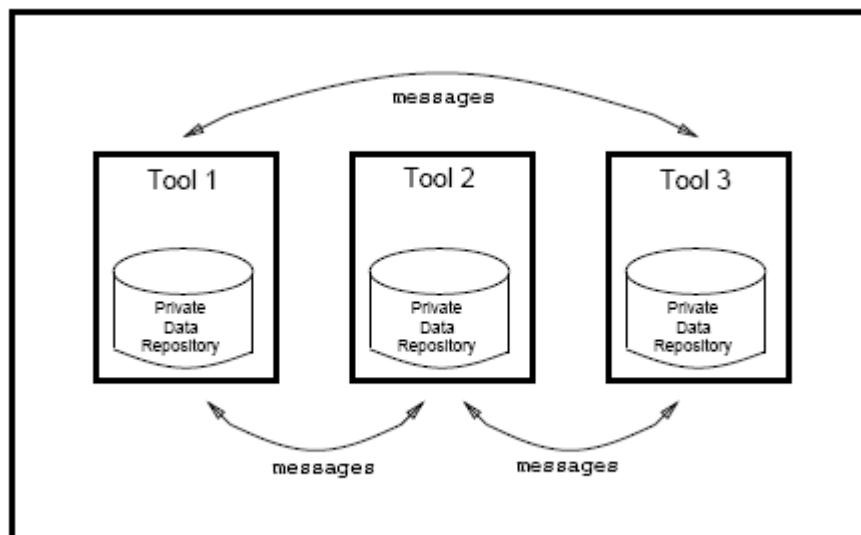
### **2.1.1. Early Message-Passing/Control Integration Approaches**

Brown, in his 1993 paper states that tool integration approaches up to the time of his writing had been focused on “data sharing”, but he suggests focus should be placed on “control integration” approach based on message passing instead [13]. In his work, he evaluates and compares three implementations named FIELD, Softbench by HP and ToolTalk by Sun. In this implementations and the conceptual model presented in his work, tools communicate using messages, which have a standard content structure. An interface for tools are developed that can communicate using

this standard message structure. Tools broadcast messages when events occur. All the tools in the system receive the messages and those with interest in these events use the message content to perform corresponding actions (See Figure 7).

The approach defined by Brown is used in ALF, which is discussed in the next section. The shortcoming of this approach is a need to develop a messaging protocol that is supported by all the tools in the system. Brown discusses this in the section “How Easy Is Encapsulation?”, and agrees on the amount of required effort.

Message passing/control integration approaches are criticized in the literature for lacking the possibility to specify functional data dependencies between complex, structured documents [30]. These dependencies are specified by data sharing approaches.



**Figure 7 Control integration/Data sharing approach [13]**

The PLETIN method proposed in this thesis does not depend on predefined messages protocols, but use the information implicitly or explicitly provided by the users instead. This information is derived from process definitions/models and is converted to a web-service definition for the specific tool. This method, if applied to a large number of cases can be used to establish a common understanding of what information users exchange with the tools. This can be used as a foundation to

construct standard domain ontology and message protocols, which most message passing/control integration approaches try to achieve.

### 2.1.2. Application Lifecycle Framework (ALF)

One of the most recent efforts on tool integration is an Eclipse project named ALF (Application Lifecycle Framework) [6]. The approach ALF takes for tool integration is "to create a multi-layered interoperability framework leveraging SOA technologies". ALF is based on the orchestration of tools to provide processes that can be repetitively executed. ALF lets the consumer control how the tools are orchestrated together [38].

Basically ALF aims to bring different tools developed by different vendors together by providing an integration infrastructure and orchestrates them to execute a process. The architecture of ALF is given in Figure 8. To be able to orchestrate and execute processes, ALF requires the tools to expose a defined set of services, i.e. be ALF-compliant. Besides, ALF requires a common vocabulary for interoperability, used to define the compliant services.

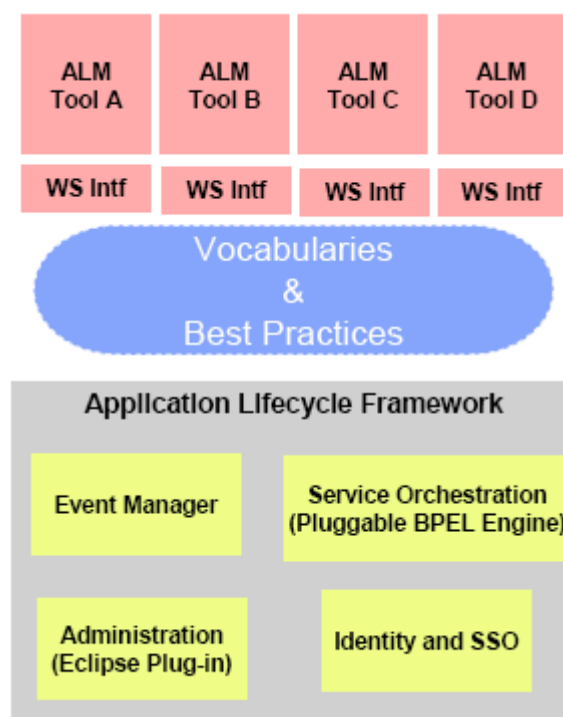
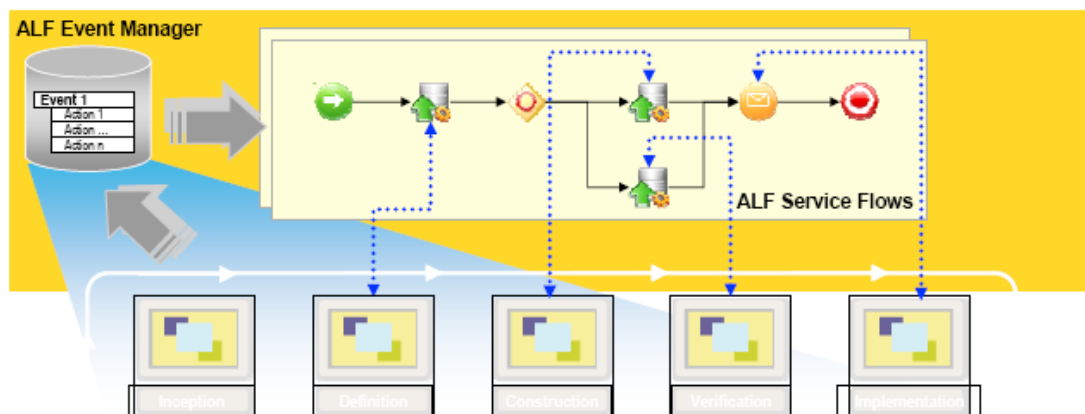


Figure 8 ALF architecture [6]

Similar to control integration/message passing approaches, every event in the ALF environment is captured by the ALF event manager. Based on the nature of these events, ALF event manager initiates pre-defined processes (called service flows) that interact with the services provided by tools (See Figure 9). The events can be generated by the actions of the users, or by other tools in response to service flow executions. This way the ALF can respond automatically to changes in the environment and integrate the tool set over these process flows and service interactions.

In short, the aim of the ALF project is to develop a standard-based tool integration environment. In this environment tools communicate using a common vocabulary. The tools are expected to provide/expose ALF-compliant services so that they can be orchestrated to execute the processes.

Unfortunately, the ALF project was terminated while it is in Eclipse incubation stage, due to insufficient community participation except from tool vendor Serena [25]. The most important phase of the project, as stated in the “Termination Review”, was the determination of “a set of domain vocabularies that define the events, objects and attributes”. The technique employed was to bring together experts from the industry, receive their opinions to start discussions and reach an agreement at later stages. However, the lack of participation from the community, and possible bias from the contribution of a single vendor resulted in the project being archived and the termination of further developments.



**Figure 9 ALF mechanism [6]**

Consulting expert opinion is a method commonly employed to understand domains. However, we believe that using data and information based on actual practices can provide a better understanding of the software engineering domain. The solution proposed by this thesis depends on organizational processes to provide a tangible basis for the integration requirements, rather than expert opinion, which is arguably abstract.

### 2.1.3. ToolNet System

Altheide et.al. in their paper [1] state, although IDEs and tool integration mechanisms have been a “hot research topic” since the beginning of 90s, there are no widely used practical solutions.

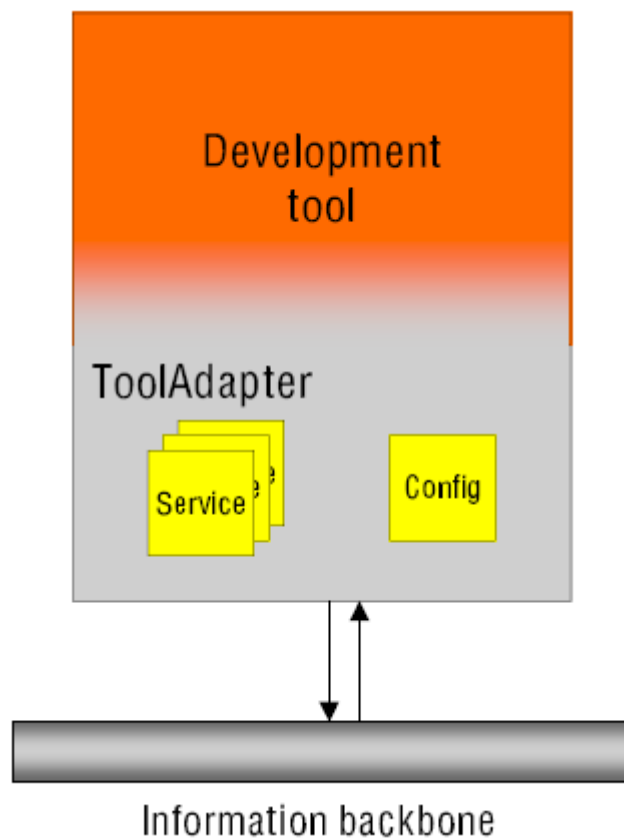


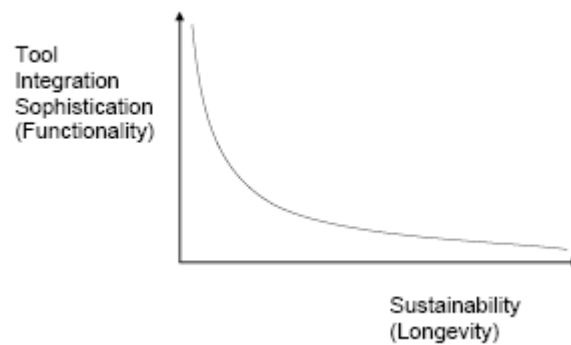
Figure 10 ToolNet architecture

To guide the development of a practical solution, they describe a sustainable tool integration mechanism that can:

1. Exchange and explore data between tools while maintaining consistency

2. Support process integration with a high degree of automation
3. Support interchangeability of tools with similar functionality
4. Easy realization of minor changes for new releases of tools
5. Employ currently used standard tools
6. Focus on integration tasks

They group existing tool integration efforts into two: use of a single repository mostly found in tool suites, and bi-lateral integration between two tools, which is widely used in practice. Focusing on the weakness of both approaches, they propose a solution where a single interface for each tool is defined connected to an integration backbone. This architecture is called the “ToolNet” architecture (See Figure 10). ToolNet is designed to be very simple, and sustainable. However to guarantee sustainability, it is designed to be simple (See Figure 11) so it cannot provide sophisticated patterns of interaction or tool-specific functionality . Rather it aims to provide a basic integration infrastructure with as much tools incorporated as possible. A service-oriented approach is proposed for reaching sustainability, extensibility to more complex functionality and interchangeability of tools.



**Figure 11 Tool integration sophistication vs. sustainability [1]**

The implementation of ToolNet has similarities with ALF and PLETIN, in which service-based adapters are used to wrap tool functionality and communicate with the environment. However, ToolNet develops adapters for specific functionality like reporting or consistency checking and ALF uses community-driven vocabularies that tools should support. On the other hand, PLETIN uses organizational processes as

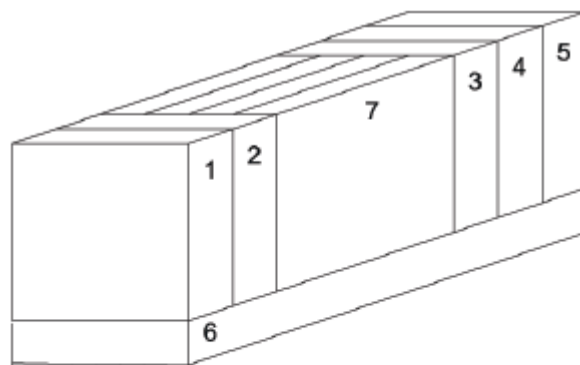


requirements to develop which functionality is required and aligns the integration infrastructure to organizational aims.

#### 2.1.4. Data-Sharing Approaches

As Brown suggested, early efforts in integration was based on a shared repository and a common understanding and definition of the domain objects. [36] suggests:

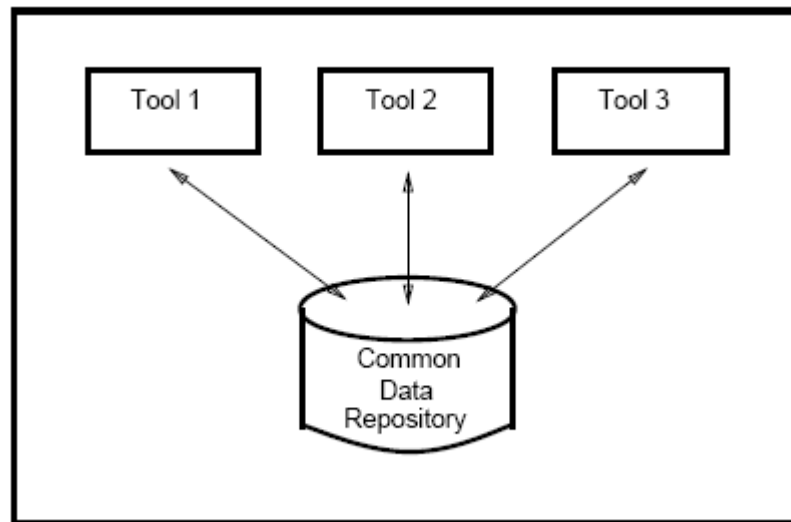
“The broadening of scope to other development concerns forced attention to be paid to support for data integration: sharing data and integrating tools with respect to the data they share. One common theme was repository-based integration, an integration model that posited a common model for the shared information and provided support for its storage and management of concurrent and secure access. PCTE [50] is a well known exemplar of this approach.”



1. User Interface Services
2. Task Management Services
3. Data Integration Services
4. Data Repository Services
5. Operating System Services
6. Message Service Network
7. Tool Slots

Figure 12 Toaster model [62]

ECMA [28] defines a reference model for frameworks of (integrated) software engineering environments. It presents a “Toaster Model” depicted in Figure 12 on how tool integration must be realized. Implementations like Portable Common Tool Environment (PCTE) followed ECMA, using the same, shared repository/database approach (See Figure 13).



**Figure 13 Data sharing approach**

This approach evolved to the development of meta-models for model exchange and resulted in standards like EIA/CDIF [29], MOF [47], XMI [66] and their various variants, due to the lack of expressiveness of the modeling techniques at the time [36]. These standards for metadata exchange are commonly used for the integration of UML-based [58] CASE tools [2]. However, integration for non UML-based CASE tools is still a challenge. All information in these tools needs to be represented using the common metadata exchange format for integration.

[35] differentiates between activities and concerns. Activities are “concrete actions and situations that take place in system development projects”. Examples are pair programming sessions, unit testing, refactoring etc. Concerns are on the other hand “what the project is really about”. Examples are analysis, design etc. Each action can contribute to more than concern. Thus:

“providing support for specific concerns is problematic. Tool integration should focus on integrating tools supporting specific activities. This leads to a requirement to integrate tools with heterogeneous data and process support.”

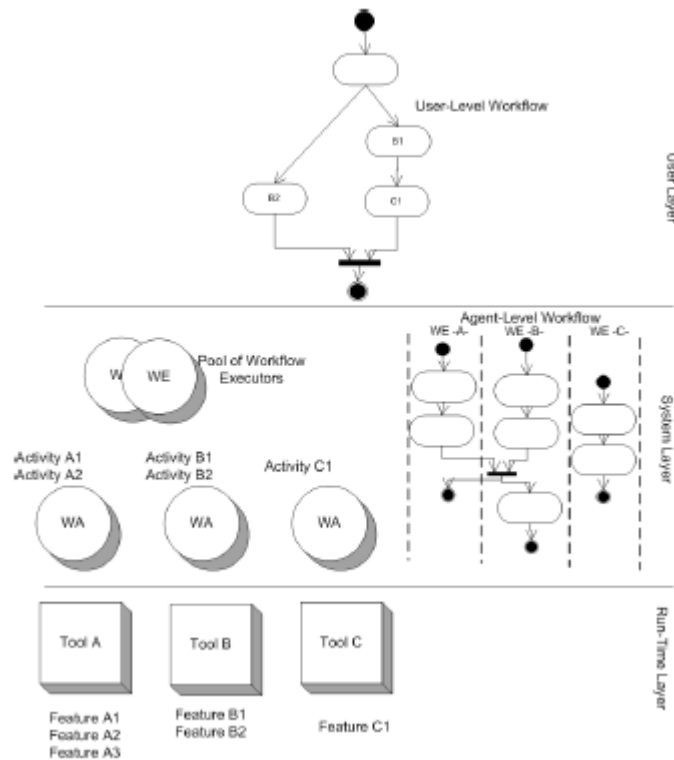
### **2.1.5. Agent-based Approaches**

Corradini et. al. propose an agent-based approach to tool integration in [22]. They propose two levels of abstraction to the complexity of tool integration. The first one

is a wrapper agent for each tool for tools interoperability, while the second one is a set of utilities used to compile workflows into agent pools.

These abstractions create a three-tier infrastructure as given in Figure 14:

1. User layer focuses on workflows
2. System layer contains the agent environment
3. Run-time layer interacts with the tools



**Figure 14 Architecture for agent-based tool integration**

The proposed solution extends UML Activity Diagrams [58] to capture workflows, which are compiled to agent activities running on an agent platform. The agents generated from the workflows interact with the wrapper agents transforming tool services to execute the workflows.

This approach presents similarities to PLETIN, where models are used to define workflows and the execution of workflows result in invocation of services provided by the tools. However the PLETIN method emphasizes the modeling effort in which the wrapper services developed as tool interfaces are defined directly from the

process models, rather than supporting generic services. This is a more process oriented approach and would result in extended functionality in terms of satisfying organizational requirements.

#### **2.1.6. Process Centered Software Engineering Environments (PCSEEs)**

Osterweil's paper [52] "posted the need for semi-automated support for the software process, in addition to tool support for artifact development". This gave rise to the development of PSEEs [36].

Barthelmeß defines PCSEEs (or Process Centered Software Development Environments, PCSDEs) as "systems that provide automated support for software development activities" [9]. According to [9] PCSEEs:

"allow for the definition and enactment of procedures performed by groups of developers working on a common project. A PCSDE stores definitions of processes in terms of steps that need to be performed, artifacts produced and transformed by these steps, of users that should perform the steps, sometimes given in terms of roles, and of constraints on execution, such as precedence among steps."

Barthelmeß [9] presents a review of PCSEEs in the literature. He describes and categorizes PCSEEs with respect to how they describe processes (coverage of descriptions) as:

1. Rule-based (MARVEL, OIKOS, EPOS, Merlin)
2. Task-based/Step-Directed (SPADE, APPL/A)
3. Artifact-Based (PROSYT, Shamus)
4. Role-Based (Pasteur, SOCCA).

He then evaluates and compares these efforts based on:

1. Latitude of interpretation (how policies are enforced, process descriptions are evolved, deviations are handled)
2. User-environment interaction

3. Inter-user communication
4. Management assessment

PCSEEs are significant for this thesis since their goal is to support and constrain the software development processes either by supplying rules or pre-defining transformations and goals (like artifacts). The approach proposed in this thesis employs process definitions to understand how tools are used in software development and provide models to dictate tool behavior. Similarly, PCSEEs define rules/graphs to guide or constrain people on how they work. A completely integrated tool set along with process guidance thus presents similarities to a PCSEE.

PCSEEs aim to support the collaborative processes, which is extremely hard since collaborative processes are characterized by “the impossibility of completely pre-defining their unfolding due to the high degree of change” [9]. The PLETIN method proposed in this thesis, however, focuses on the menial tasks performed by users to maintain cooperation of tools, in other words tool integration. The repetitiveness of these tasks renders them perfect candidates for formal description and automation contrary to the challenges collaborative tasks provide.

## **CHAPTER 3**

### **THE PLETIN METHOD**

The PLETIN (shorthand for Process L<sup>E</sup>vel Tool INtegration) method is a four-stage method developed to identify and then implement integration-tuples from process definitions in a software development environment.

PLETIN is developed based on a case study conducted as a part of this thesis. It has been developed iteratively during the conduct, and new findings were applied recursively to steps already completed whenever necessary.

The PLETIN method is based on the scenario where there are no integrations between the tools in a software development environment, cooperation of the must be maintained manually. As an example, to make it possible for different tools that are not integrated to work on the same data set, the data must be fed to each tool manually. Similarly, for a tool to operate on the information created by another tool, data should be moved between the tools by a user manually. In other words, users must perform actions necessary to keep the tools working together (cooperate). In this situation, process definitions (or models derived from these definitions if they exist) would contain sequences of actions (what we name integration-tuples or sequences, and use interchangeably in this thesis) performed to maintain tool integrations. These sequences are required to keep the tools working cooperatively. Thus, user actions account for non-existing integration facilities of the tools. For simplicity, we call these facilities integration-tuples (or sequences). An integration-

tuple is a candidate tool integration situation. In our scenario users maintain integration-tuples manually. A method designed to investigate the process models can be used to understand how users interact with the tools to maintain these tuples. Based on this knowledge, requirements for the services to support these actions can be inferred and implemented to build a tool integration framework.

The first stage of PLETIN is called the *context definition* stage, where the scope is defined. In this stage software development processes for which tool interactions are either non-existent, constrained to a single interaction or inherently complex are excluded. The scope can be defined by direct examination of process definitions or models. This information is usually already available to the software engineering process group (SEPG) that has developed (or is developing) the process definitions. Organizing a meeting with the process group, or inclusion of an experienced process group member in the scope meeting can help exclusion of process definitions that provide insufficient information for further work. This stage uses process definitions (or process models) as inputs and outputs a list of processes that is going to be examined further in the later stages of the method.

Process components are identified based on the scope and represented on a formal process model in the *process definition* stage. In this stage user interactions with tools are analyzed to uncover candidate tool integration situations. A process model is developed for each process definition to visualize the interactions with tools. *Process definition* stage uses the scope identified in the first stage as input and produces process models visualizing tool interactions as outputs. Tool interactions that satisfy certain criteria are labeled as sequences (integration-tuples). These sequences are mapped to existing services or APIs provided by the tools to develop an integration infrastructure.

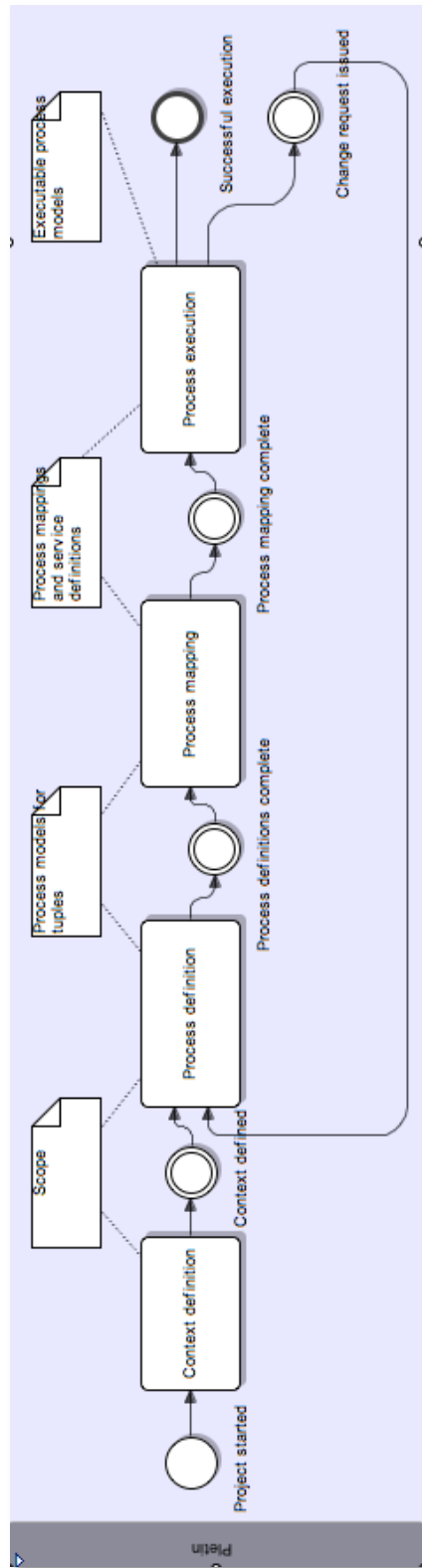


Figure 15 Process model for PLEtin



A process model including mappings between actions and services is developed in the *process mapping* stage. The aim of this stage is to understand how users interact with tools and to develop services that can respond to the actions performed by users. This stage of PLETIN uses process models developed in the previous stage as input and produces a detailed description of atomic actions performed by users on tools and services that can respond to these actions as outputs. Actions and services are combined into a business process that is represented as a process model. This business process can be executed on a business process execution engine.

These models are deployed on a process execution engine for actual implementation. Process executions are monitored and necessary feedback for process change is developed in the *process execution* stage. The process model for PLETIN is given in Figure 15.

PLETIN is a tool integration technique based on organizational process definitions, rather than data interchange formats or ad-hoc standardization frameworks. As stated by [52] “software processes are software too”. Tools provide automation facilities for specific sub-processes of software development. Integration of these separate “islands of automation” would result in a more complete and continuous execution of software development. Thus, the tool integration effort must be treated like software too. PLETIN aims to develop an understanding of user interactions with tools to build requirements necessary to develop an integration framework. The implementation approach is based on Service Oriented Approach (SOA), where fragments of systems are connected together using a standard based framework. Using PLETIN, organizations can integrate existing toolsets based on the requirements generated by their own processes. In the long run, an industry-wide understanding of requirements for tool integration can be developed. These requirements can be employed by, or even forced upon vendors to develop standard-based, interoperable, interchangeable tools supporting software development processes.

PLETIN requires the existence of and is based on process definitions. So the quality of its outputs is directly correlated to the quality of process definitions. If process definitions are not available in an organization, it would be much more beneficial to combine a process definition/modeling/improvement effort with the execution of the PLETIN method.

### **3.1. Modeling Approach**

Assuming every activity (except those performed internally by individual tools) in software development is performed manually and there is no integration between the tools used, consider the following: a user would like to cooperate some tools. To achieve this goal, he is required to perform a sequence of successive, simple operations on different tools, moving data between them.

An example for such a sequence is: “Team Leader creates a baseline in requirements management tool, named <projectName>-YYYY-MM-DD. He then creates a build label in software configuration management tool, with the same name as the baseline”. It is clear from the example that there is a sequence of two actions on two different tools performed to maintain cooperation of different tools for a common goal. Some more generic examples are:

- Create Data1 on ToolA, create Data1 on ToolB.
- Read Data2 on ToolC, create Data2 on ToolD.
- Update Data3 on ToolE, delete Data3 on ToolF.

These sequences of actions that are performed by users on different tools, hint the existence of candidate integration situations between tools. The user merely cascades changes or moves information to another tool. Such mundane tasks are very good candidates for automated execution, and they can be executed through integration implementations.

Knowledge on candidate integration situations can be used as requirements for interoperable, interchangeable tools. Tool designs can incorporate interfaces/services that can satisfy the requirements presented by the processes employed in software

development organizations. These requirements are derived from the knowledge on candidate integration situations. On a more practical level, these can be realized into actual implementations through the use of business process execution environments.

PLETIN is a method to identify and optionally realize possible integrations between different software engineering tools. PLETIN can either use existing process definitions of the organization or can be executed in parallel with a process definition/modeling effort. PLETIN presents guidance for the process of converting process definitions into service definitions that can be used as requirements to develop custom interfaces for the tools. These correspond to integration implementations.

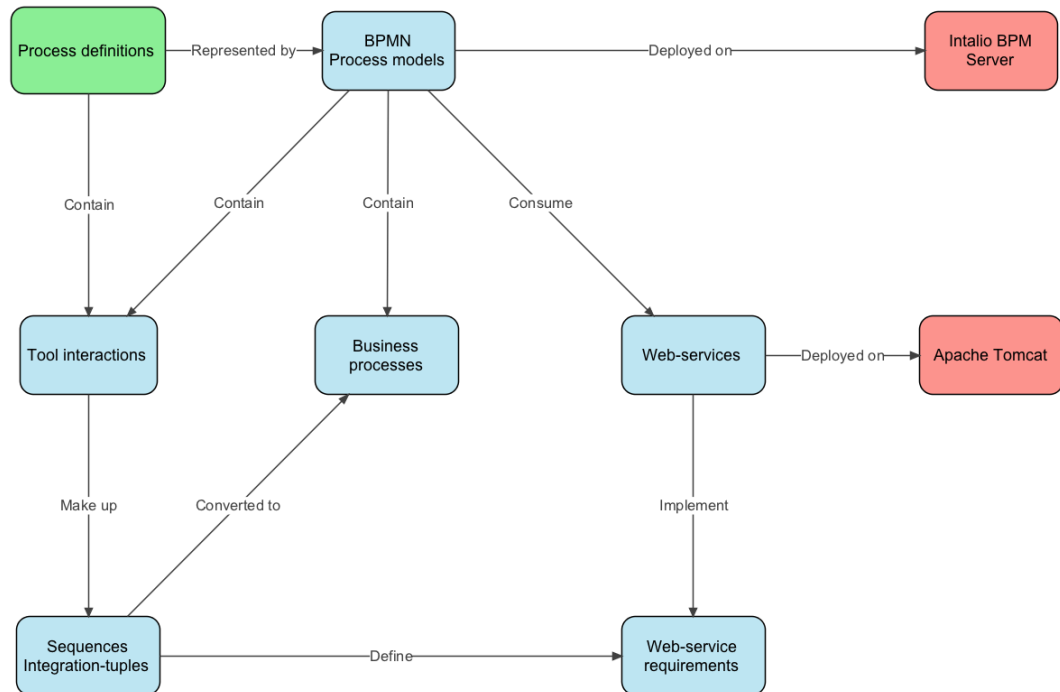
The approach of PLETIN is based on the identification of sequences of tool interactions in user processes. Sequences satisfying criteria for the number of tool interactions and complexity and type of user interaction are chosen. These sequences are treated as candidate integration situations between different tools.

PLETIN identifies user actions that contain tool interactions. To be able to identify those interactions, “users”, “manual user actions”, “tools”, “user interactions” with these tools, and “messages” sent and received between these components are discovered from process definitions. The relationships between these components are developed into process models. Sequences of tool interactions are identified from process models. These interactions are then later implemented in a process execution environment. A conceptual map for the terms used by the PLETIN method is given in Figure 16.

PLETIN looks for “simple” user actions that contain tool interactions to identify candidate tool integration situations. For this, inspection of only certain processes is required. Not all processes in software development contain such interactions. PLETIN does not demand the analysis of processes and actions that don’t have tool interactions or only have a single interaction throughout the complete process.

It should also be noted that sequences of actions that are classified as complex can have complex data mappings, demand decision-making and even creative

capabilities. For the scope of this work, such interactions are left out of scope since the implementation and even the definition of them may require substantial effort.



**Figure 16 Conceptual map for the PLETIN method**

In the context of this thesis, a tool interaction is subjectively classified as simple if it is a CRUD (Create, Read, Update, Delete and Execute) operation. The definition is similar to those used in persistent storage or database systems. See Section 3.10 for a detailed discussion.

To decrease the effort and time spent applying the method, scope should be defined. In the context of PLETIN, the scope is defined such that only the processes with multiple, yet simple, user-tool interactions are included. This information can be directly obtained from the process group in a meeting, or revealed through an inspection of process definitions. More detail is given on the specific activities on the Section 3.3.

In this work, Business Process Modeling Notation (BPMN) [17] is used as the modeling notation and Intalio|BPM Community Edition [37] is selected as the

process design, deployment and execution environment. Intalio|Designer is the process modeling component of Intalio BPM Suite used to develop BPMN models. Further discussion on the selection of the modeling notation is available in Section 4.1.1. Since the PLETIN method can co-exist with concurrent process modeling work, it can be modified to use another modeling notation if the notation supports the representation of the following required components: users, tools, manual user actions, tool interactions and messages. The actual implementation of integrations also require certain functions from the underlying platform like the ability to execute processes directly from process models, easy/one-button deployment, data mapping and process instance monitoring. Intalio|BPM Community Edition used for this work provides these features out-of-the-box. A detailed description of the BPM Suite is given in Section 4.2. However, any process modeling environment providing process execution facilities similar to importing web-service definitions, direct invocation of web-services, data mapping can be used for the purpose with slight modifications.

### **3.2. Method Stages**

PLETIN is designed so that it is executed in a software development organization and monitored and improved continuously. With PLETIN, an integration framework for the tools is laid out according to the requirements set by actual processes of the organization. Based on this integration framework, tools or the integration implementations can be changed at later stages. This brings inter-operability to tools.

PLETIN has four stages. During the *context definition* stage, the scope of the modeling process is defined. Based on the scope, process components are identified and represented on a formal process model in the *process definition* stage. In this stage user interactions with tools are analyzed to uncover candidate tool integrations situations. Tool integrations that satisfy certain criteria are labeled and mapped to existing services or APIs provided by the tools to develop an integration infrastructure. A process model including mappings between actions and services is developed in the *process mapping* stage. These models are deployed on a process execution engine for actual implementation. Process executions are monitored and

necessary feedback for process change is developed in the *process execution* stage. The process model for PLETIN is given in Figure 15.

Each stage of the method is explained in the subsequent sections and process models using BPMN notation defining each stage are presented.

### 3.3. Context Definition (Stage I)

The first stage of PLETIN defines the scope of the effort. In this stage those processes where tool interactions during software development are either non-existent, constrained to a single interaction or inherently complex are excluded.

**Table 1 Types of process with respect to the number of tool interactions**

<b>Type of process</b>	<b>Process definition contains</b>	<b>Information provided</b>
Type 0	No tools	No tool interactions possible, no integration opportunities.
Type I	Single tool Single interaction	A single tool does not present an integration opportunity. At least two tools are required.
Type II	Single tool Multiple simple interactions	Although a sequence of interactions on a single tool does not present an integration situation it is of interest from an automation perspective.
Type III	Multiple tools Multiple simple interactions	Multiple tool interactions may present integration situations if interactions are simple.
Type IV	Only complex interactions	Complex interactions cannot be represented adequately on process models, and executed

The promise of PLETIN is the fact that successive user interactions with multiple tools constitute candidate integration situations between the tools. Thus the basic requirement of PLETIN is the existence of successive interactions with one or more

tools in process definitions. Process definitions having no tool interaction can't provide any information on possible integrations. A process definition having a single tool resembles the case where there are no tools since interactions with a single tool can't provide information on possible integration. However, multiple "simple" interactions with a single tool might provide automation opportunities instead of integration situations. The scope of the effort can be defined to include such processes if automation is one of the primary goals of the effort. Table 1 represents the types of processes and the information they provide.

**Table 2 Sample process list for RE process area**

<b>Process Code</b>	<b>Process Name</b>	<b>Tools</b>	<b>Process Type</b>
RE51	Preparation	RM, SCM	III
RE5211	Elicit needs	SCM	I
RE5212	Establish customer requirements	RM	I
RE5213	Review customer requirements	RM, SCM	III
RE5214	Validate customer requirements	RM, SCM	III
RE5221	Establish software requirements	UML, RM, SCM	III
RE52211	Define product components and interface requirements	RM, TT, SCM, UML	III
RE52212	Establish software requirements	RM	II
RE52213	Review software requirements	RM, TT, SCM, UML	III
RE5222	Validate software requirements	RM, SCM	III
RE531	Manage changes to requirements and inconsistencies between requirements and work products	RM, SCM	III

Following the information in Table 1, the scope is defined to include processes of Type III. Optionally, processes of Type II can also be included. All other types of processes are excluded. The definition of scope can be achieved by direct examination of process definitions or models. This information is usually already available to the process group that has developed (or is developing) the process definitions. So organizing a meeting with the process group, or inclusion of an experienced process group member in the scope meeting can help exclude process definitions providing no information for further work. A “process list” in the form of a table including process name, process code, the tools used in the process and the process type (Type 0, I, II, or III) is sufficient for filtering and future reference. A sample table is given in Table 2.

As depicted in Figure 17, the stage begins with the project initiation. Process modelers organize a meeting with the Software Engineering Process Group (SEPG) to get information on the process definitions. If SEPG does not exist or is not available, this information can be extracted from process definitions. However, SEPG can provide the information faster and more accurately.

In this stage, evaluating criteria like process execution frequency, average error rate during manual process execution and user feedback on the process nature (repetitiveness) is beneficial. This information can be used to prioritize the analysis and possibly implementation of process definitions. This provides larger benefits to be reaped earlier. The data can be appended to the “process list”.

### **3.4. Process Definition (Stage II)**

In the context definition stage processes that are suitable for the application of the method are selected for further analysis. In process definition stage, processes including multiple tool interactions are analyzed to extract information on candidate tool integration situations. Process components including actors, actions, process flow, tools and messages are identified. Details on the identification of each component are given in the next subsections.



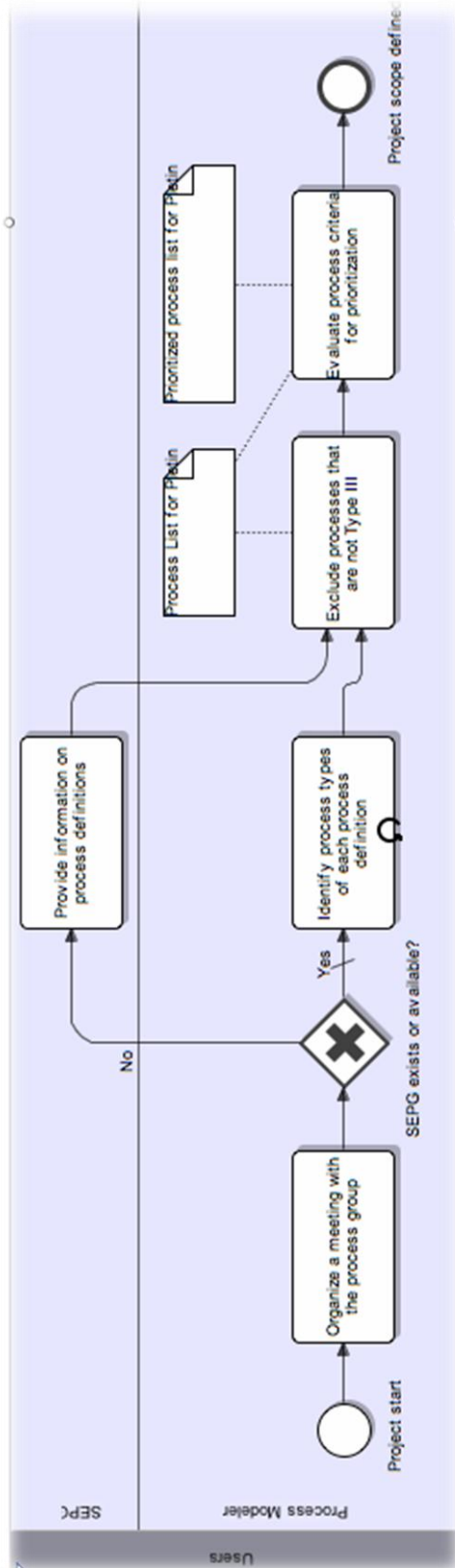
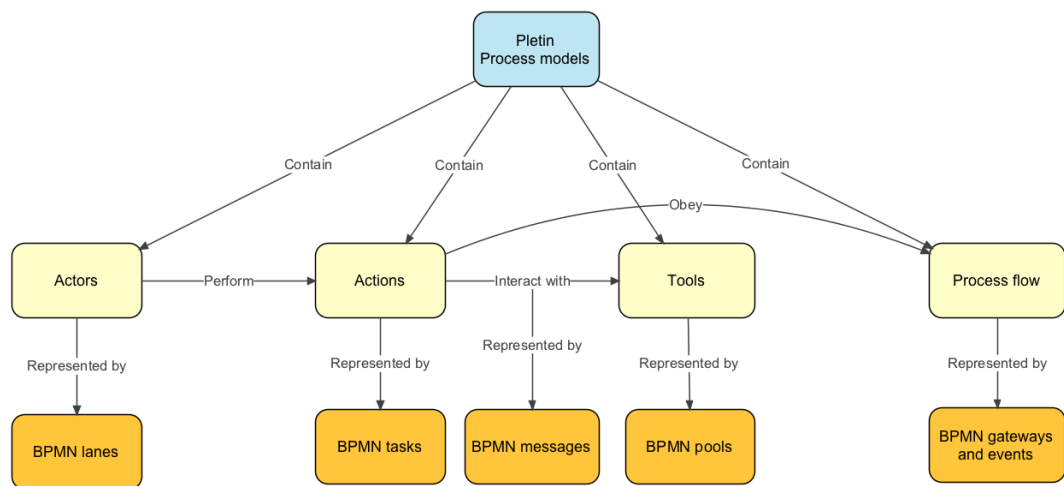


Figure 17 Process model for PLETIN Stage I, Context Definition

The identified components are represented on a process model for visual analysis and identification of tool interaction sequences. The complete manual process is represented as a single BPMN pool in a BPMN diagram. This BPMN pool is marked “not-executable” since the actions are performed manually. All actors are represented as individual BPMN lanes in this BPMN pool. Actions are represented as BPMN tasks assigned to actors. Process flow is represented using BPMN gateways, BPMN events and the flow of tasks. Each software development tool is represented in a separate BPMN pool.



**Figure 18 Relationships of PLETIN BPMN Elements**

They are depicted as external to the software development process so that the required interfaces are visible. The internal processes of the tools are left outside of the scope of this modeling effort. So the pools for the tools are represented as empty, as a “black box”.

The actions that have interactions with tools are connected to BPMN pools that represent tools. The connections are done with BPMN message elements. They are used to identify information exchanged with the tools. The relationship between BPMN elements used during the implementation of PLETIN is given in Figure 18.

Actions with simple tool interactions within the model are highlighted because the method requires the identification of them. Actions with complex interactions are left as they are.

The next step is to identify highlighted interactions that are successive. If there are more than one action that have simple interactions with tools (so that they are highlighted in the model) executed in succession, this group of actions are identified as a sequence. A sequence is represented on the process model as highlighted actions grouped together using the “BPMN Group” element. Process model for the process definition stage of PLETIN is given in Figure 19.

Figure 23 is a part of a process model, presenting a sequence of two simple successive actions constituting a sequence. This sequence is highlighted using “BPMN Group” element. A sample process model created in this stage is given in Figure 22.

### **3.4.1. Actor and Action Identification**

Every step taken to achieve a goal in a process definition is an action. Actions are usually described in single sentences. The subject who performs the action is noted as the actor [9]. The verb and the object define the action. An example is as follows: “Team Leader creates a baseline in requirements management tool.” In this example, Team Leader is the actor because he is the one that performs the action. “Create a baseline” is the action, performed by the actor.

The PLETIN method represents each actor identified from the process as an individual BPMN lane in a common BPMN pool. This pool is labeled “<ProcessCode>-PeopleProcess”, where <ProcessCode> is to be substituted by the unique identifier of the process under analysis, for example “RE5214”.

This pool contains lanes for all the roles taking part in the process. These roles perform actions and interact with the software development tools. A sample representation as BPMN lanes of four different users (DTM, TL, PMA, Customer) participating in a process is given in Figure 20.

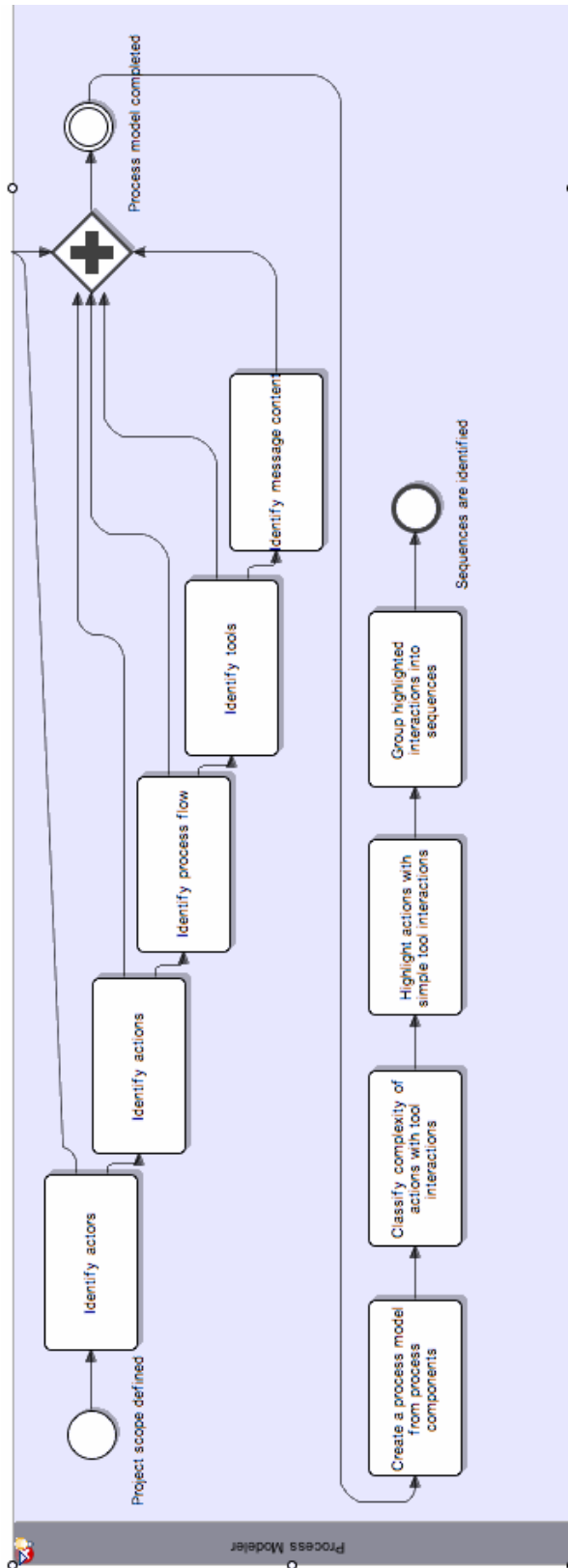
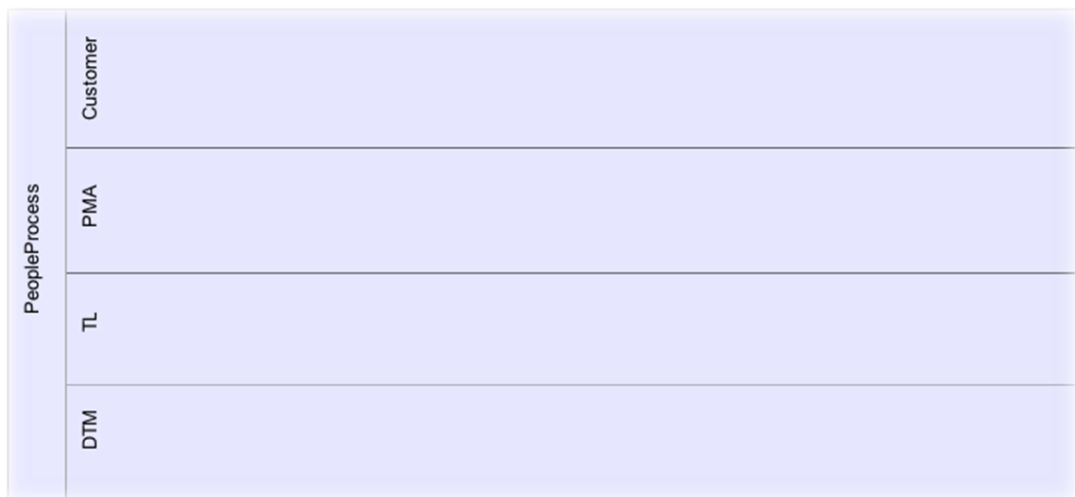


Figure 19 Process model for PLETTIN Stage II, Process Definition

### 3.4.2. Process Flow Identification

Process flow defines in what order the tasks are executed. It includes the following information:

- Start/End conditions of a process
- Sub-processes
- Task dependencies
- Parallel task execution
- Process branching and merging
- Intermediate events and conditions during process execution



**Figure 20 Sample User Representations on Process Model**

This information is extracted from the process definitions and represented as a BPMN process model through the use of BPMN constructs like BPMN gateways, BPMN events and BPMN tasks. A sample process flow is presented in Figure 21. Detailed description of BPMN constructs is available in section labeled BPMN .

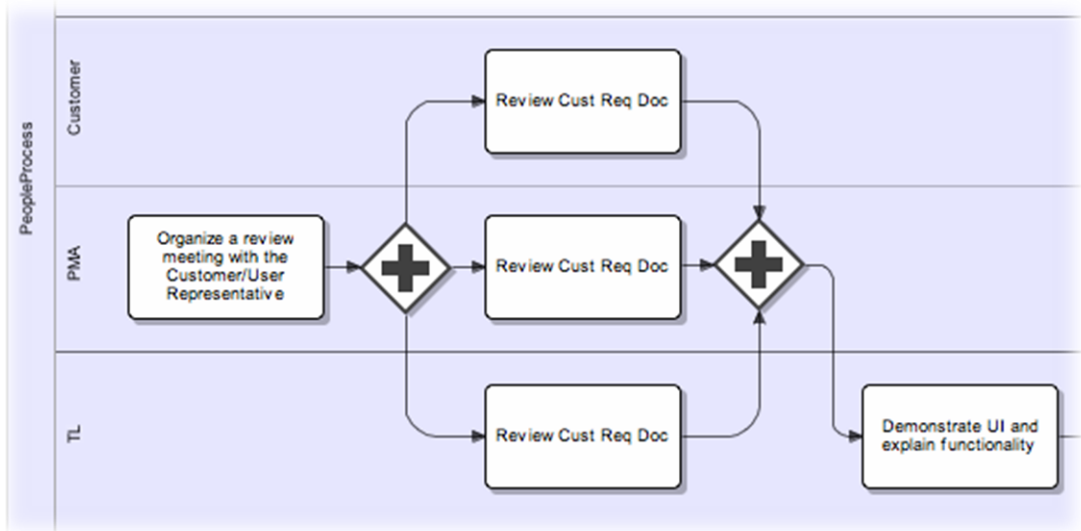


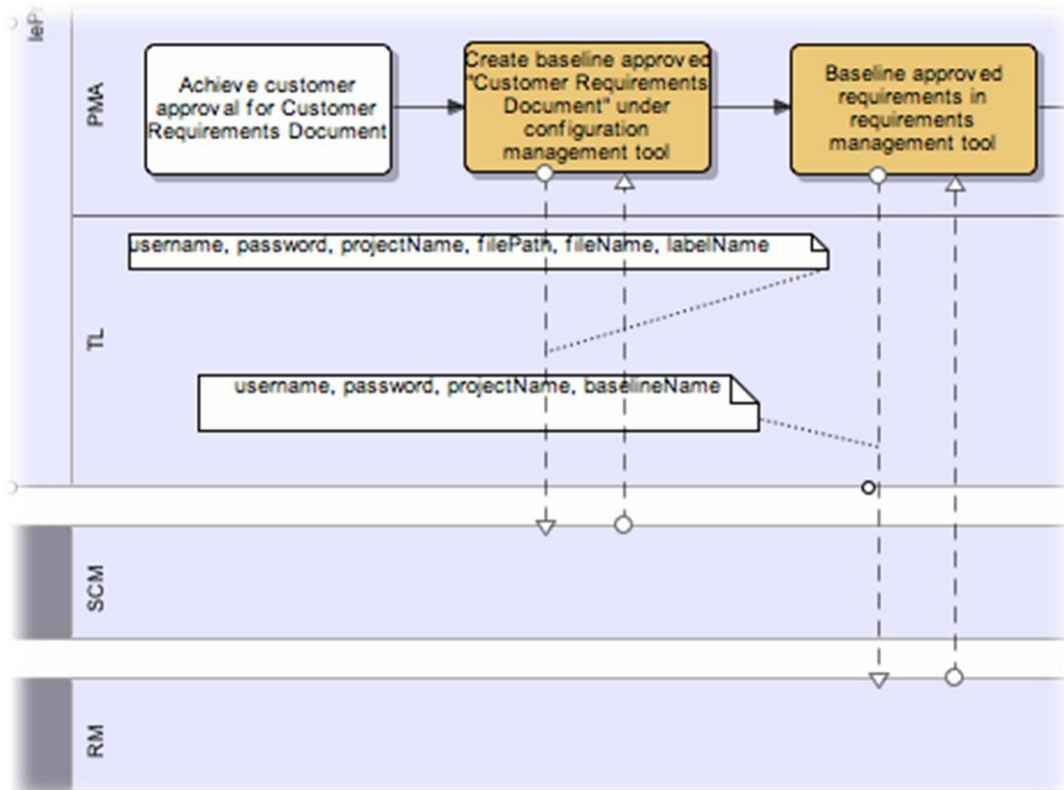
Figure 21 Sample process flow with BPMN notation

### 3.4.3. Tool and Tool Interaction Identification

Tools are represented as independent, empty BPMN pools in process models. They are marked as “not-executable” since the tool itself executes the actions. An empty, not-executable pool provides a black-box perspective in the process model. While the interface for the tool interaction is clearly visible, the complexity of inner-tool operation is hidden from the process model.

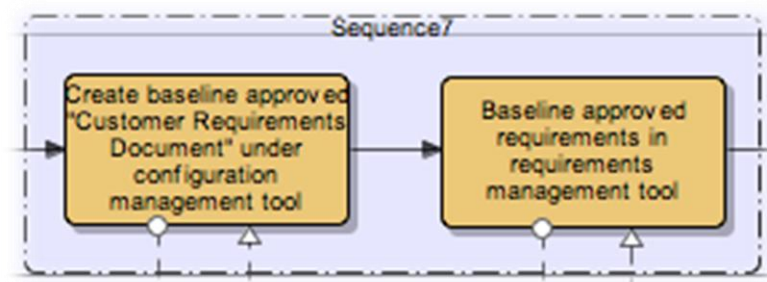
Every tool in the process model must be connected to a task with BPMN messages. These messages represent the requests made by the user to the tool and the responses provided by the tool. The messages are connected to “BPMN data objects” to represent the content of the messages, which are critical for PLETIN. The message contents are used to determine the input/output parameters required for the implementation of integration.

Tool interactions are classified whether they are complex or not. An action is classified as simple if it is one of the CRUD operations: Create, Read, Update, Delete or Execute. Simple tool interactions are highlighted with a distinctive color (e.g. orange) on the process model. If these interactions make up a sequence there exists a candidate integration situation. A sample for the representation of tool interaction in a process model is given in Figure 22.



**Figure 22 Sample tool interaction represented as a process model**

The highlighted tool interactions are grouped together using BPMN group objects to represent a sequence. A sample of such grouping is given in Figure 23. These sequences, consisting of multiple, simple tool interactions present candidate tool integration situations. Their structure and interfaces are identified in the next stage of PLETIN, process mapping where necessary information for developing custom integrations is developed.



**Figure 23 Two actions grouped into a sequence**

### **3.5. Process Mapping (Stage III)**

Process definition stage outputs a process model similar the one given in Figure 22. Sequences of user interactions are represented on the model. The information captured by this process model is used as an input for the process mapping stage.

The process flow for the process mapping stage is given in Figure 24. It is the most critical stage in PLETIN. The aim of this stage is to understand the details on how users interact with tools to develop actual implementations that can interact with the tools on behalf of the users.

#### **3.5.1. Identification of Atomic Actions**

Each action in the identified sequences is broken down to atomic actions performed by the user. An atomic action represents the smallest, indivisible unit of action a user performs while interacting with a tool. Examples of atomic actions include authentication, file checkout, command issue etc. Observing a user performing processes can unveil atomic actions easily.

“BPMN sub-process” objects are placed on the model to substitute the actions highlighted in the previous stage, representing simple tool interactions. The highlighted actions are broken down to atomic actions. Atomic actions are also represented as BPMN tasks. They are placed inside the BPMN sub-process corresponding to the action they are created from. Thus, every highlighted action identified in the previous stage is replaced with a sub-process including its atomic actions. A sample breakdown is given in Figure 25.

For each atomic action in the model, a corresponding “placeholder” BPMN task is created in the BPMN pools representing tools. These placeholder tasks in tools represent the services tools should provide. Based on the relationships represented on the process model, actual services are developed that are going to substitute these tasks. The atomic actions and placeholder tasks of the tools are connected by BPMN message elements to represent the interface required for the integrations. A sample is given in Figure 28.



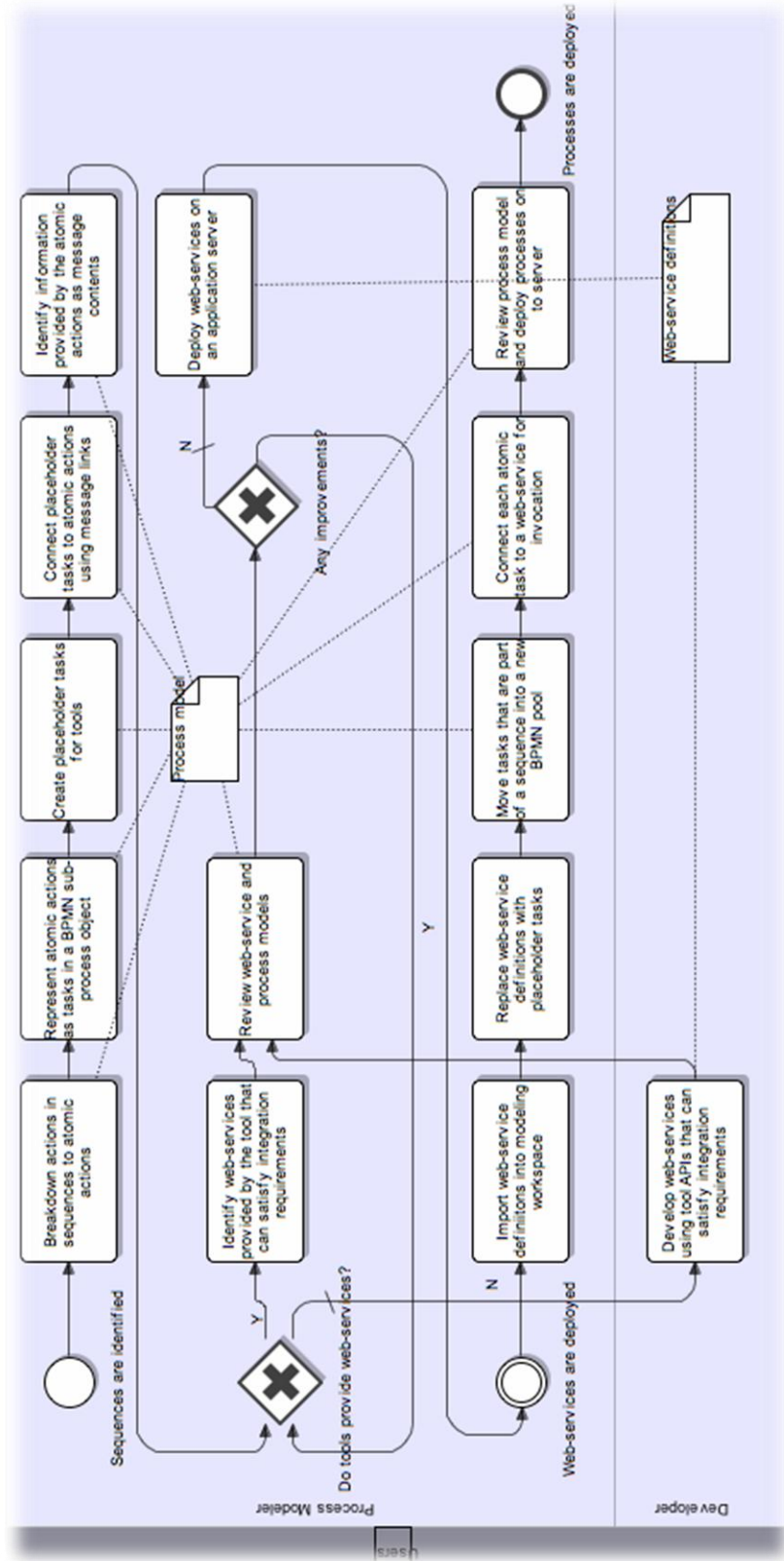
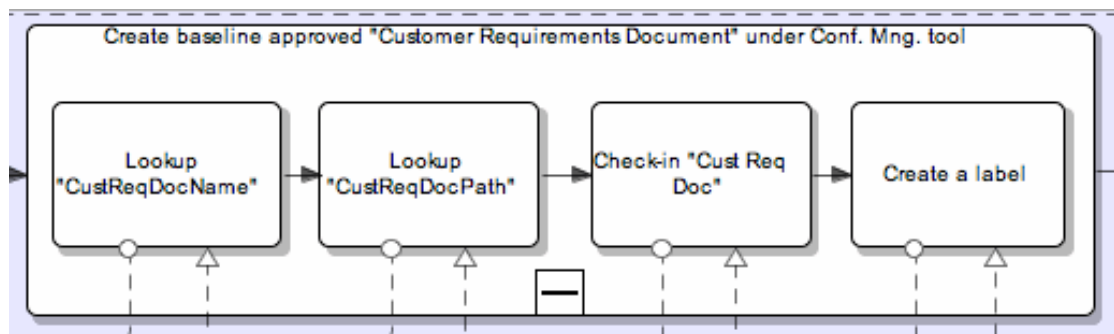


Figure 24 Process model for PLETTIN Stage III, Process Mapping

We should note that, the portion of the process model represented in Figure 28 is the final version of the atomic actions. It has been modified after the initial atomic action definitions are analyzed and requirements are compared to the existing facilities provided by the tools. The details of the comparison activity are given in the next section on the development of custom integration implementations. It is recommended that actual implementations be based on a web-services infrastructure; however, this constrains how the information is handled during execution. Since complex objects like “session” or “connection” are not transportable in a web-services environment, information on how and where to login is embedded into every action (tool interaction) for the final version of the atomic action definitions.



**Figure 25 Sequence breakdown**

The implicit and explicit information provided by the atomic actions are noted in the process model as BPMN data objects connected to BPMN messages. The responses generated by the tools are also recorded. Since there are many messages passed in even simple process definitions, recording message contents on the process model itself may introduce clutter. A better approach would be to record this information in a separate location, like a spreadsheet. This information is used as the requirements for integration implementations.

### **3.5.2. Identification of Implicit Sequences**

The approach PLETIN uses is to analyze process definitions to identify candidate integration situations in software development. The integration situations are

uncovered in the form of "sequences" of actions users normally perform manually to maintain tool cooperation.

However there are cases where a candidate or existing tool integration may not manifest itself as a sequence in process models, but remain hidden. These are called "implicit sequences". Users of the method would get aware of implicit sequences if a very well known integration situation or a tool is not visible in the outputs of the method.

**Table 3 Types of implicit sequences**

<b>Type of Implicit Sequence</b>	<b>Method of identification</b>
Interrupted	a. Observe data flows (process artifacts) b. Examine Submit/Update/Put actions
Compound	a. Examine existing integration maps, tool features b. Look for mentions of two or more tools in integration definitions
Unmentioned/Omitted	a. Examine existing integration maps, tool features b. Improve process definitions by observation
Complex	Change interaction complexity decision criteria

There are 4 types of implicit sequences. Types of implicit sequences and methods for their identification are given in Table 3. Details are as follows:

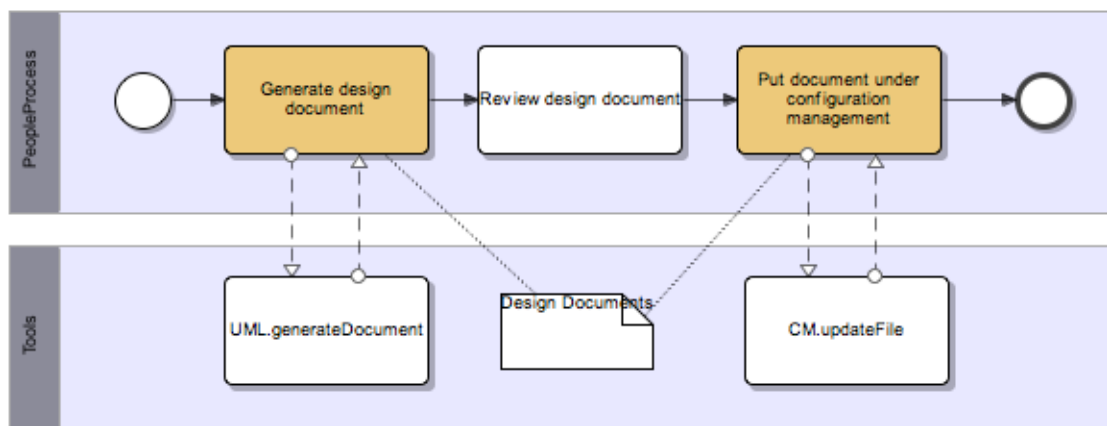
### **Interrupted Implicit Sequences**

Two simple tool interactions separated by a single manual/complex action (by definition) don't constitute a sequence. Only simple tool interactions in succession are considered as such. In this case, even if this interrupted sequence were a valid candidate integration situation, it would not be detected by PLETIN.

Interrupted implicit sequences are observed when a software development artifact like a document is generated using a tool and submitted to another tool after one or

more manual operations. A common example is the submission of a document generated by a tool to configuration management system, only after it has been reviewed and accepted. In this case, generate document and submit actions are simple tool interactions. However, they are separated by a manual review and approval process, which prevents the construction of a sequence. The link between these two actions is the document (the process artifact) employed by both actions. A sample representation for an interrupted implicit sequence is given in Figure 28.

Interrupted implicit sequences can be uncovered by giving special consideration to data flows in process models. Process models can be modified to include BPMN Data Objects representing software development process artifacts like documents. This information can be obtained from process definitions during Process Definition stage of PLETIN. Another simpler approach would be to pay special attention to Submit/Update/Put actions where information/data is provided by the action to a tool. These types of actions require a source for the information they are providing. The source could be the output of a tool interaction. If that is the case, the decomposition of the Submit/Update/Put action into its atomic components, a sequence of interactions can be identified.

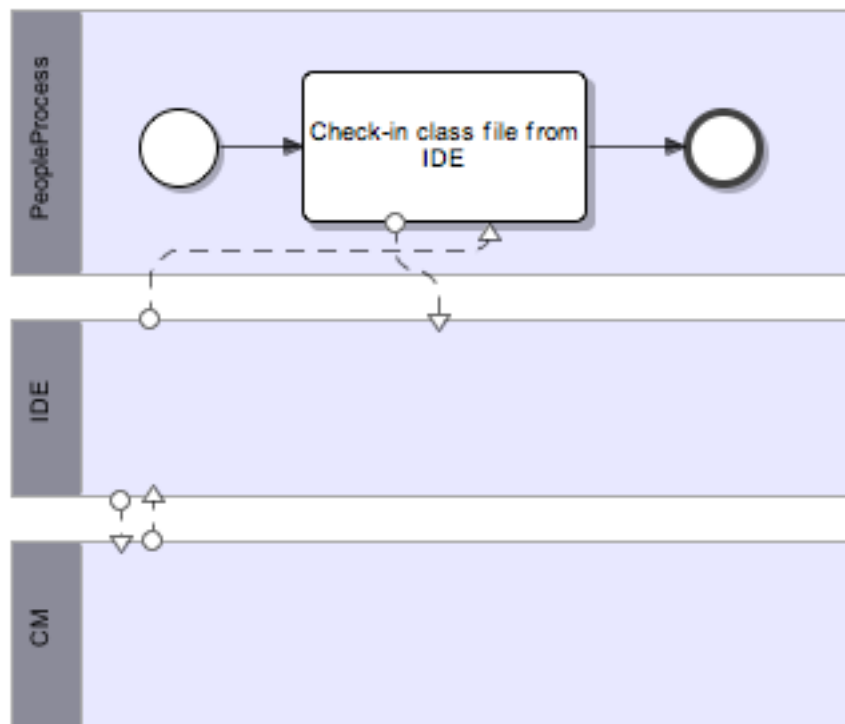


**Figure 26 A sample interrupted implicit sequence**

However, users of the PLETIN method should be aware of the fact that, changing the order of actions in a process definition can change the output of the process. This may have unintended consequences in process execution.

## Compound Implicit Sequences

A compound implicit sequence occurs if a tool interaction by a user triggers a tool-tool interaction. In other words, when a user performs an action that results in the tool interacting with another tool, the second interaction would not be visible in the process model depicting the interaction. Compound implicit sequences are observed if there are already existing integration implementations between tools and they are already employed by the organization. In this case, the integration is considered as a functionality of the tool and is not explicitly described in the process definition or subsequent process model. A sample compound implicit sequence is represented in Figure 27.



**Figure 27 A sample compound implicit sequence**

Compound implicit sequences can be uncovered by analyzing existing integration maps (See Figure 47 for an example). These maps represent already existing integration implementations between tools. Existing integration implementations that are not identified as candidate integration situations by PLETIN must be analyzed and depicted as process models by decomposing the user interaction into several

interactions as if the user is maintaining the integration manually. Although existing integration maps may contain information on this type of implicit sequences, actual process execution must be sought for, to produce correct process models. Another alternative is to look for tool interactions in process definitions mentioning more than two tools.

It should be noted that, existing integration implementations already employed by the organization might be omitted from process definitions. This type of implicit sequences is discussed below. Thus, identification of compound implicit sequences may result in improved processes for the organization.

### **Unmentioned/Omitted Implicit Sequences**

PLETIN uses process definitions developed in organizations as input. The quality of PLETIN's outputs depends on the quality of its inputs. Any omissions in these process definitions will result in missed/undetected integration opportunities. If the process definition effort by the organization prior to the implementation of PLETIN had omitted/unmentioned some of the tool interactions, sequences of these interactions would not be detected by PLETIN.

This type of implicit sequences is the hardest to remedy because they are inherently missing from method inputs rather than being implicit. Existing integration maps can be used to uncover unmentioned implicit sequences, similar to compound implicit sequences. Observation of actual process execution is another approach. However, actual observation may not give better results than existing process definitions because it is prone to the same problems.

### **Complex Implicit Sequences**

During the Context Definition stage of PLETIN, each tool interaction performed by users is classified to be simple or complex. For a more detailed discussion on the context definition phase of PLETIN and classification of actions with respect to complexity, see Section 3.3. Only simple interactions are highlighted on process models. Sequences are constructed only from simple interactions. Thus, a sequence

consisting of complex interactions would not be included in the scope of the method and would not be visible in the outputs of the method.

Complex implicit sequences are artificially introduced by the subjective identification of simple tool interactions during the context definition stage of PLETIN. Changing the decision criteria to implement more complex interactions would result in the detection of this type of implicit sequences, since complex interactions would instead be labeled as simple.

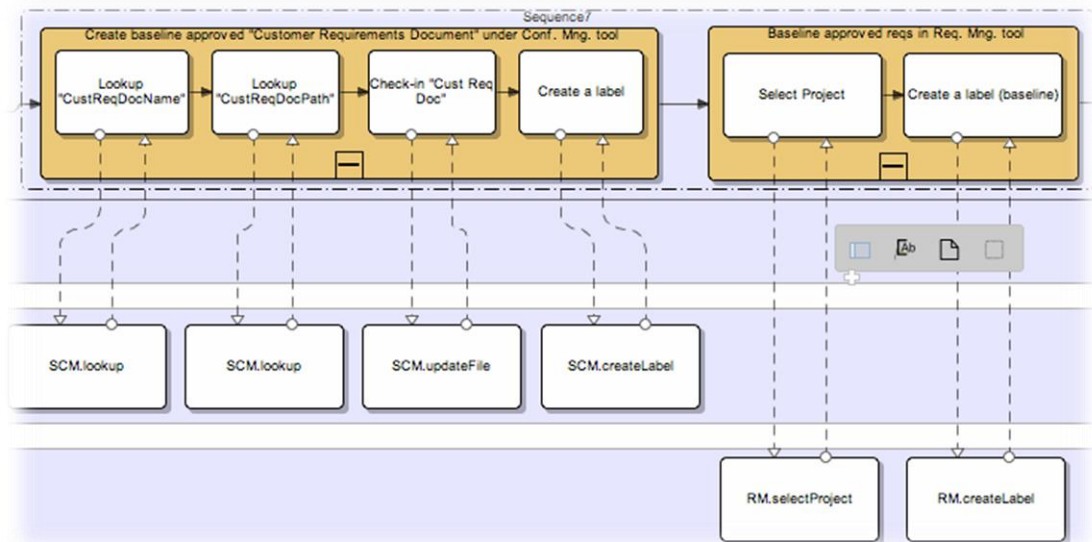
### **3.5.3. Development of Custom Integration Implementations**

The placeholder tasks represent the services tools should provide. They define draft web-service definitions that will respond to the user actions. A descriptive name is given to each placeholder task. These names are later used as the name of the web-service. The name should also specify the tool type. If tools already provide web-services conforming to the interface specifications defined in this process model then they can directly be used. However, if no web-services are provided, custom implementation that satisfies the integration requirements is necessary.

During custom development, web-services are designed so that they replace the placeholder tasks. Inputs and outputs required for the web-service design are already available on the process model. They are shown as input and output messages for the placeholder tasks.

Existing web-services or APIs provided by the tools are investigated to develop web-service implementations that can fulfill the requirements. Web-services are implemented using these facilities provided by the tools, or from scratch. Top-down, and bottom-up service identification techniques similar to those employed in SOA service definition methodologies are used [7].

Preliminary web-service definitions and process models are reviewed for reuse opportunities and improvement. For example “login” operation can be and should be embedded into each service definition because complex objects like “session” and “connection” cannot be transported using SOA messaging protocols. This review can result in changes on both the web-service definitions and process models.



**Figure 28 Placeholder tasks for tools**

After the web-service definitions and process models are reviewed and any inconsistencies are resolved, web-service implementations are created and deployed to an application server of choice. Process modelers can perform web-service implementation if the group has familiarity with the concepts. However, it would be more beneficial if external support can be obtained in the form of experienced software developers from the organization, or from vendors.

Web-service definitions in the form of WSDL files are imported into Intalio|Designer workspace. Intalio|Designer enables its users to import WSDL files directly from a network location and then add these web-service definitions to process models by drag-and-drop. Regular BPMN tasks are used to invoke these web-services. Web-service definitions are inserted in the model as tasks, substituting the placeholder tasks created in the identification of atomic actions stage.

A new BPMN pool is created for each sequence in the process model and labeled as “<ProcessCode>Seq<SequenceNumber>”. An example is: “RE5214Seq7”. This helps easy identification of the sequences both in process models, and during actual process execution. Tasks that are grouped as a sequence, converted to a sub-process and broken down into atomic actions are moved into this new BPMN pool. This pool is marked as “executable” since its contents will be executed on a business process



execution engine to perform actions on behalf of the users. Tasks not labeled as part of a sequence are left in the people process pool. A BPMN “start message event” is added to the beginning of the new pool. Also, a BPMN “end message event” is added to the end of the process in the pool. A new task is created that invokes the start message event, named “Invoke sequence <SequenceNumber>”. This task is a replacement for the sequence removed from the people process pool. A BPMN message connection is created between this new task “Invoke sequence <SequenceNumber>” and the “start message event” in the pool representing the sequence. The end message event is also connected to the task “Invoke sequence <SequenceNumber>” using a BPMN message connection to send a response after the process is being executed. An example for this task is visible in Figure 29.

BPMN pools representing different tools can be combined into a single BPMN pool called “<ProcessName>-Tools” or can be left independent. All pools representing tools should be marked as “not-executable”.

Each atomic step in a sequence invokes a corresponding web-service. These web-service representations created from WSDL files are imported into Intalio|Designer workspace. BPMN message connections are used for invocations. Data mappings are created so that information input by the user and information returned by tools as web-service responses are routed to correct places. For a sample data mapping, see Figure 32. An example process model with web-service invocations is available in Figure 29.

When the process is complete and error-free it is deployed to the Intalio|Server.

### **3.6. Process Execution (Stage IV)**

Processes deployed to the Intalio|Server are reviewed for consistency and completeness. Test runs are performed and execution is monitored. Suitable changes are implemented if necessary.

After all requested changes are applied to process models, process models are accepted to be consistent, and it is validated that they conform to manual execution of processes in practice; users can start executing processes that employ custom

implementations for integration. The “people process” portion of process models developed by the PLETIN method visually represents how users can execute manual processes. It is used for process communication in the organization.

Sequences extracted from the process definitions were converted to separate, more detailed description of how users perform the actions in the process mapping stage. These sequences of actions are replaced by a single task that expects the user to invoke a sequence. Users can logon to Intalio|BPM Community Edition web interface to execute the sequence by selecting it from the list of processes. They are required to provide information necessary for the execution of the process. This way, by selecting the suitable sequence and providing information, users initiate the sequence that performs actions on behalf of them. The execution of the sequence is completely invisible to the user. Users are not concerned with which tools are used, which actions to perform or even which documents to handle. While providing an integration framework for different tools, PLETIN also provides partial automation for software development processes.

However it should be noted that as organizations, processes are subject to change. Thus, the integration infrastructure must be able to sustain this change and the PLETIN method must be applied iteratively to new or changed (improved) process definitions to create new or modify existing integration situations. The PLETIN method, once implemented must be executed in an iterated matter indefinitely.

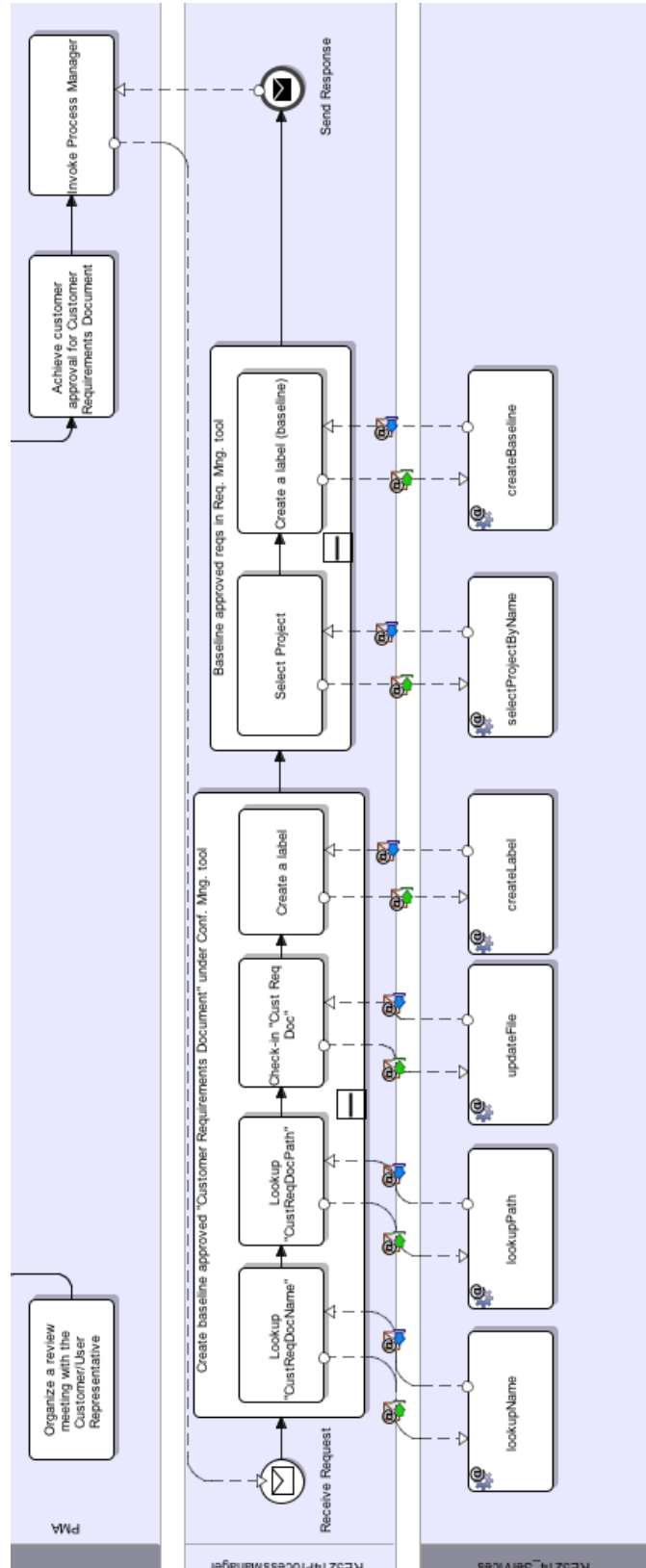


Figure 29 Web-service invocations by the process manager

### **3.7. Roles**

The following is a brief description of each role that takes part in the execution of the PLETIN method:

1. **Process Modelers:** This role is responsible for the application of the PLETIN method. A process modeler uses process definitions or process models to identify process components including actors, actions, tools, process flow and messages. These components are used to identify simple tool interactions, which are grouped and labeled as sequences/integration-tuples. Sequences constitute the requirements for the development of web-services that will be consumed by processes executed by an execution engine to perform tasks on behalf of users. The process models this role develops result in the development of the tool integration framework. Process modelers are required to be familiar with process modeling, have an understanding of the organization and software development in general. It will be beneficial if modelers are familiar with the BPMN notation, BPM techniques and SOA technology in general. Process modelers are not required to develop web-services and integration implementations if there are people with development expertise in the project staff. However, during the process mapping stage, process modelers are required to develop web-service definitions and validate the implementations. A project manager should manage the efforts of the process modelers. They should work together with the SEPG to obtain detailed information on process definitions, resolve ambiguities and identify inconsistencies. Support from SEPG on the “context definition” stage would provide helpful to quickly identify processes with no tool interactions.

During our research, a single person conducted the case studies and method implementation. The number of process definitions, their complexity and the ease of access to information can be used to determine the number of process modelers suitable for the project size.

2. SEPG: SEPG is a group of people responsible for the definition and management of processes in an organization. Since the PLETIN method requires extensive information on processes, SEPG support for the process modelers are crucial for the success of the project.
3. Developers: Developers are external to the execution of the method. However during the “development of custom integration implementations” step in process mapping stage, modelers can benefit from the experience of this role to develop integration implementations faster and better.
4. Users: End users are not involved in the application of the PLETIN method until test runs of the processes. After the processes are verified and deployed, end users logon to Intalio|BPM Community Edition web interface to initiate the execution of the processes.

### **3.8. Notation**

During the development of PLETIN and the case studies, BPMN was used for process modeling. Detailed information on BPMN is available in Section 4.1. BPMN provided facilities to represent all the information required for the visual representation and further execution of the processes. Thus there was no need to modify standard BPMN notation. However, the following two conventions were developed for PLETIN:

1. Tasks in process definitions with simple tool interactions were highlighted (with a color of modelers ‘choice) for easy identification. An example is available in Figure 22 and Figure 23.
2. Highlighted tasks executed in a sequence are grouped together using BPMN group objects to represent sequences in process models. An example is available in Figure 23.

### **3.9. Comparison of the PLETIN Method with Previous Efforts**

The development of PLETIN was inspired by the approach taken by ALF (See Section 2.1.2). ALF is a control integration effort, where events are captured from

the environment and corresponding service flows are executed. PLETIN, on the other hand, depends on users initiating processes in a specific step of the manual process.

PLETIN presents similarities to past control integration efforts discussed in Section 2.1.1 where a message passing method is used. In PLETIN, messages exchanged between users and tools, and between tools are identified and re-created by web-services and encapsulated in web-service technologies.

PLETIN also have similarities to ToolNet System discussed in Section 2.1.3. ToolNet System develops an interface for all the tools in the environment, called the ToolNet Adapter. ToolNet Adapter supports the least common denominator functionalities so that tools can communicate with each other. ToolNet's approach is to provide sustainability by providing a small subset of functionality over a large number of tools. Although this may work in simple scenarios, support for organizational processes would definitely suffer.

Our research showed that, although much research has been done for data sharing approaches discussed in Section 2.1.4, none has been widely accepted. We believe that the reasons behind this are as follows:

Data-sharing approaches require a complete perspective on the software development processes. However, information on every aspect of software development, including common data representations and common message formats are not available.

Data-sharing approaches are not sustainable. A new tool, a new functionality of a tool or a new conceptual representation requires the definition of new common representations compatible with the existing ones. This is not feasible.

Data-sharing approaches are not suitable for existing tool sets, where an arbitrary selection of tools are available. In such an environment developing a common data representation presents significant challenges.

In their paper [57] Thomas and Nejme details the types of integration developed by Wasserman. Discarding the platform integration type, they discuss the details of presentation, process, control and data integration types. Regarding the aspects of

tool integration put forward by Thomas and Nejme, we can say that PLETIN supports the following:

1. Control Integration
  - a. Provision (to what extent are a tool's services used by other tools in the environment?)
  - b. Use (to what extent does a tool use the services provided by other tools in the environment?)
2. Process Integration
  - a. Process step (how well do relevant tools combine to support the performance of a process step?)
3. Data Integration
  - a. Interoperability (how much work must be done for a tool to manipulate data produced by another?)

PLETIN also partially supports the following aspects, if these concerns are included in the process definitions used:

1. Process Integration
  - a. Event (how well do relevant tools agree on the events required to support a process?)
  - b. Constraint (how well do relevant tools cooperate to enforce a constraint)
2. Data Integration
  - a. Non-redundancy (how much data managed by a tool is duplicated in or can be derived from the data managed by the other?)
  - b. Data consistency (how well do two tools cooperate to maintain the semantic constraints on the data they manipulate?)
  - c. Data exchange (how much work must be done to make the non-persistent data generated by one tool usable by the other?)
  - d. Synchronization (how well does a tool communicate changes it makes to the values of non-persistent common data?)

PLETIN uses process models to identify and describe user integrations with tools. This approach is also taken by PCSEEs (See Section 2.1.6) in the literature, where generally software development processes are represented as process models. However, PCSEEs convert these process models to rules and constraints, represented in specialized languages. These rules and constraints are used to guide and constrain the process flow. PLETIN does not transform the models, but include more detail on the interactions to develop automatically executable versions of the user tool interactions to develop tool integration implementations.

In their paper, Mi and Scacchi [46] suggest process models should be used to realize integration. They state that interfaces between the tools can be derived from process models, as proposed in this work. They also focus on an existing integrated toolset, and provide process flexibility. However, contrary to these approaches, PLETIN focuses on the identification of candidate integration situations. PLETIN aims to understand “when to integrate?” and “what tools to integrate?” to support “which processes?”. PLETIN takes a process-oriented approach to find out which integration functionality will be the most beneficial for the execution of the processes. We haven’t been able to find out such an approach in the literature.

Briefly, PLETIN presents a process-focused approach to tool integration, which provides practical benefit to organizations with already existing tool sets. We believe that these are the missing aspects of tool interaction hindering popular adoption of frameworks in practice.

### **3.10. Limitations of PLETIN**

The following is a list of areas that have been identified as the limitations of the approach proposed in this thesis. These can be improved by further research and the use of newer technologies.

1. The PLETIN method uses existing process definitions and process models as inputs. It does not aim to define, or improve organizational processes. To apply PLETIN in an organization without process definitions or in an organization planning for process improvement, it is recommended that a



process definition/modeling effort is completed, or PLETIN is implemented parallel to such an effort. An extension to PLETIN, that analyzes process evidence (e.g. tool logs) to develop process definitions can be developed and employed in organizations without process definitions.

2. The PLETIN method discards all user interactions with tools that are classified other than Create, Read, Update, Delete and Execute. This is a subjective and arbitrary constraint and can be improved by further formalisms, more complex mappings and more complex representations of the nature of the actions.
3. During the development of PLETIN an assumption was made suggesting all tools are “grey boxes” to the modelers [59]. A “grey box” tool means although the source code is not available for the tool, an API or an extension language is provided [31], [32]. PLETIN in its current form is still useful for “white box” tools where the tool is custom developed or open sourced. However, in case of a “black box” tool where the modelers have access to only binary executables, then an enveloping approach where a wrapper converting internal tool objects to necessary format is required [59].
4. It is not possible to resolve “implicit sequences” (see Section 3.5.2) with re-organization using PLETIN. Such re-organization is considered as part of a process improvement and left out of scope of this thesis
5. The PLETIN method is developed and applied using BPMN as the process modeling notation and Intalio|BPM Community Edition for implementation. Use of other notations and execution infrastructures were not considered, however they could provide valuable insights on data and process representation.
6. The application of PLETIN method in many organizations across software development industry would create a knowledge base of integration requirements and a basic understanding of the components of an ontology including objects, actions, roles and messages. Development of an ontology

was left outside the scope of this thesis, but it would definitely help a better understanding of the software domain.

7. PLETIN in its current form executes processes specifically initiated by the users from the web interface. In a software development environment, events generated by other tools, or external sources are not taken into account. However, this is easy considering that PLETIN uses a BPEL engine, which can capture external events and initiate corresponding processes.
8. Complex mappings of data between tools were not accounted for in the scope of this thesis. It can be implemented as a new tool (similar to the project repository developed for the first case study), or the facilities provided by the business process execution engine can be employed.
9. In the case of already existing (legacy) but inferior integration solutions or new integration functionality delivered with a new release of the tool, PLETIN can be modified to employ or discard the existing functionality. However, the comparison of a solution custom developed based on the PLETIN method and an existing solution is not in the scope of this thesis.

## CHAPTER 4

### ENABLING TECHNOLOGIES

#### 4.1. Business Process Modeling Notation (BPMN)

BPMN is a standard modeling notation initially developed by Business Process Management Initiative (BPMI). Object Management Group (OMG) [50] currently maintains the standard since the two organizations merged.

BPMN aims to provide a notation [17]:

“that is readily understandable by all business users, from the business analysts that create the initial drafts of the processes, to the technical developers responsible for implementing the technology that will perform those process, and finally, to the business people who will manage and monitor those processes”.

BPMN, having multiple target user groups is simple, yet sufficiently expressive. Both high-level manual processes and low-level automated processes can be modeled using the notation.

Another goal stated in the BPMN Specification Version 1.1 [17] is to ensure BPMN can be used to visualize languages designed for the execution of business processes,

such as BPEL4WS (Business Process Execution Language for Web Services) [16], later renamed to WSBPEL [64].

The design of BPMN was preceded by the review of other notations like UML Activity Diagram, UML EDOC Business Processes, IDEF, ebXML BPSS, Activity-Decision Flow (ADF) Diagram, RosettaNet, LOVeM, and Event-Process Chains (EPCs) to consolidate best ideas from these [17].

#### **4.1.1. Choice of BPMN**

BPMN has been chosen as the notation used in this thesis for the following reasons:

1. BPMN is designed from the ground-up to model both manual and automated processes. The BPMN specification does not provide a direct mapping between BPMN and execution languages like BPEL4WS or WSBPEL. However, there exists a significant overlap between BPMN and BPEL4WS constructs and many tools supporting BPMN modeling provide facilities for converting BPMN to execution languages. Modeling and execution of business processes in this thesis was implemented on Intalio|BPM Community Edition (See Section 4.2). Intalio|BPM Community Edition provides direct-deployment of BPMN models as executable business processes on Intalio|BPM Community Edition. The modeling workspace provided by Intalio, Intalio|Designer enables modelers to include information like data-mapping and web-service invocation.
2. Ease of visually representing executable business processes using BPMN enables the implementation infrastructure in this work to employ web-services. Web-services are a standards-based method of communication for applications of different platforms. They are widely accepted and provide a direct solution for cooperating different tools of different vendors and platforms.
3. BPMN's aim to support all business users lets users develop models on both ends of the detail spectrum. Modelers can create high-level models involving entities external to the organization, use black-box perspectives to identify interactions and create people-oriented visual representations of manual




process definitions. Using the same notation, detailed models of processes to be executed automatically can be developed. This dual modality is critical for the aim of this thesis, because the PLETIN method aims to understand user interactions with tools from models representing manual processes, to develop an integration framework that can execute these interactions on behalf of the users automatically. Such an undertaking requires a flexible notation that can support both high-level people processes and low-level automatic execution of processes.
















































































4. BPMN is maintained by OMG, who also develops the UML specification. It is an open, widely accepted and actively developed standard. This assures the relevancy and continuity of the BPMN specification as a business process modeling standard. Using open and standard components for integration efforts is critical for its acceptance.
5. The last but not the least is our familiarity with the notation.

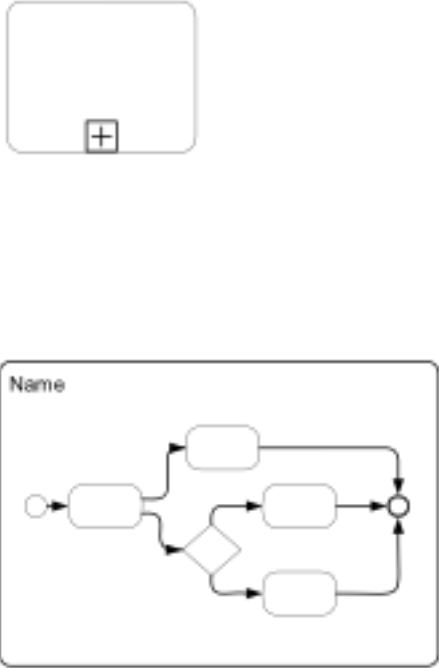
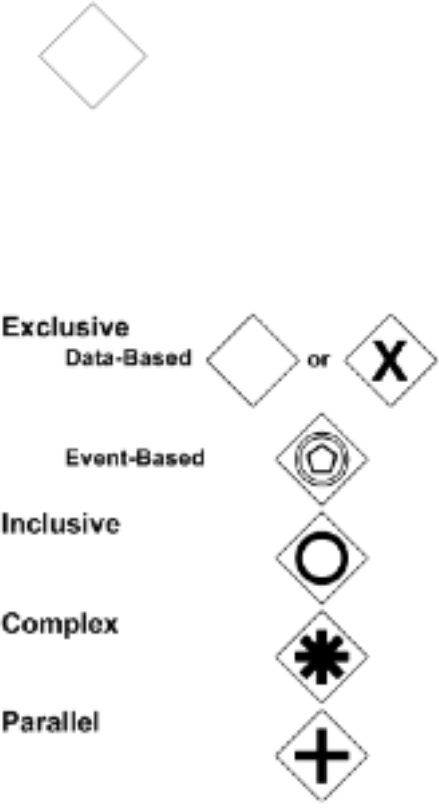
#### 4.1.2. BPMN Elements

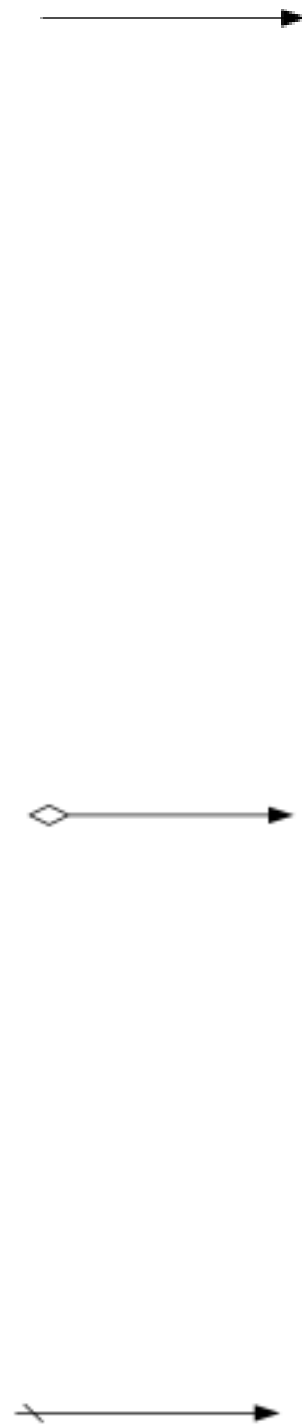
BPMN is used for business process modeling. It has common elements with other process modeling notations. Table 4 gives a list of elements supported by BPMN. This information is from the BPMN 1.1 Specification [17].

**Table 4 BPMN elements**

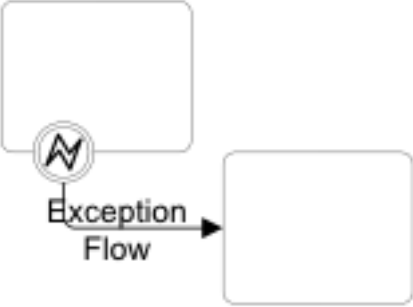
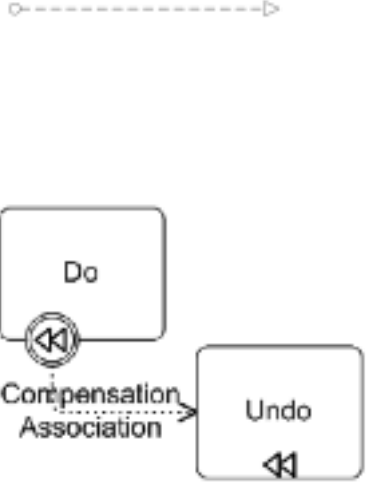

<b>Element</b>	<b>Description</b>	<b>Notation</b>
Event	An event is something that “happens” during the course of a business process. These events affect the flow of the process and usually have a cause (trigger) or an impact (result). Events are circles with open centers to allow internal markers to differentiate different triggers or results. There are three types of Events, based on when they affect the flow: Start, Intermediate, and End.	<p>Start </p> <p>Intermediate </p> <p>End </p>






	<p>As the name implies, the Start Event indicates where a particular process will start.</p> <p>Intermediate Events occur between a Start Event and an End Event. They will affect the flow of the process, but will not start or (directly) terminate the process.</p> <p>As the name implies, the End Event indicates where a process will.</p> <p>Start and most Intermediate Events have “Triggers” that define the cause for the event. There are multiple ways that these events can be triggered. End Events may define a “Result” that is a consequence of a Sequence Flow ending. Start Events can only react to (“catch”) a Trigger. End Events can only create (“throw”) a Result. Intermediate Events can catch or throw Triggers. For the Events, Triggers that catch, the markers are unfilled, and for Triggers and Results that throw, the markers are filled.</p>	<table border="0"> <thead> <tr> <th></th> <th colspan="2">“Catching”</th> <th colspan="2">“Throwing”</th> </tr> </thead> <tbody> <tr> <td><b>Message</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Timer</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Error</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Cancel</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Compensation</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Conditional</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Link</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Signal</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Terminate</b></td> <td></td> <td></td> <td></td> <td></td> </tr> <tr> <td><b>Multiple</b></td> <td></td> <td></td> <td></td> <td></td> </tr> </tbody> </table>		“Catching”		“Throwing”		<b>Message</b>					<b>Timer</b>					<b>Error</b>					<b>Cancel</b>					<b>Compensation</b>					<b>Conditional</b>					<b>Link</b>					<b>Signal</b>					<b>Terminate</b>					<b>Multiple</b>				
	“Catching”		“Throwing”																																																						
<b>Message</b>																																																									
<b>Timer</b>																																																									
<b>Error</b>																																																									
<b>Cancel</b>																																																									
<b>Compensation</b>																																																									
<b>Conditional</b>																																																									
<b>Link</b>																																																									
<b>Signal</b>																																																									
<b>Terminate</b>																																																									
<b>Multiple</b>																																																									
<p><b>Activity</b></p>	<p>An activity is a generic term for work that company performs. An activity can be atomic or non-atomic (compound). The types of activities that are a part of a Process Model are: Process, Sub-Process, and Task. Tasks and Sub-Processes are rounded rectangles. Processes are contained within a Pool.</p> <p>A Sub-Process is a compound activity that is included within a Process. It is compound in that it can</p>																																																								

	<p>be broken down into a finer level of detail (a Process) through a set of sub-activities.</p> <p>The details of the Sub-Process are not visible in the Diagram. A “plus” sign in the lower- center of the shape indicates that the activity is a Sub-Process and has a lower- level of detail.</p> <p>The boundary of the Sub-Process is expanded and the details (a Process) are visible within its boundary. Note that Sequence Flow cannot cross the boundary of a Sub-Process.</p>	
<p><b>Gateway</b></p>	<p>A Gateway is used to control the divergence and convergence of Sequence Flow. Thus, it will determine branching, forking, merging, and joining of paths. Internal Markers will indicate the type of behavior control.</p> <p>Icons within the diamond shape will indicate the type of flow control behavior. The types of control include:</p> <ul style="list-style-type: none"> <li>• Exclusive decision and merging. Both Data-Based and Event-Based.</li> <li>• Inclusive decision and merging.</li> <li>• Complex -- complex conditions and situations.</li> <li>• Parallel forking and joining.</li> </ul> <p>Each type of control affects both the</p>	

	incoming and outgoing Flow.	
Sequence Flow	<p>A Sequence Flow is used to show the order that activities will be performed in a Process.</p> <p>Normal Sequence Flow refers to the flow that originates from a Start Event and continues through activities via alternative and parallel paths until it ends at an End Event.</p> <p>Uncontrolled flow refers to flow that is not affected by any conditions or does not pass through a Gateway. The simplest example of this is a single Sequence Flow connecting two activities. This can also apply to multiple Sequence Flows that converge on or diverge from an activity. For each uncontrolled Sequence Flow a “Token” will flow from the source object to the target object.</p> <p>Sequence Flow can have condition expressions that are evaluated at runtime to determine whether or not the flow will be. If the conditional flow is outgoing from an activity, then the Sequence Flow will have a mini-diamond at the beginning of the line (see figure to the right). If the conditional flow is outgoing from a Gateway, then the line will not have a mini-diamond (see figure in the row above).</p> <p>For Data-Based Exclusive Decisions or Inclusive Decisions, one type of flow is the Default condition flow.</p>	 <p>The diagram shows three horizontal arrows representing different types of sequence flows. The top arrow is a simple arrow with a solid black arrowhead. The middle arrow has a small diamond shape at its tail and a solid black arrowhead. The bottom arrow has a diagonal slash at its tail and a solid black arrowhead.</p>



	<p>This flow will be used only if all the other outgoing conditional flow is not true at runtime. These Sequence Flow will have a diagonal slash will be added to the beginning of the line.</p> <p>Exception Flow occurs outside the Normal Flow of the Process and is based upon an Intermediate Event that occurs during the performance of the Process.</p>	
<p><b>Message Flow</b></p>	<p>A Message Flow is used to show the flow of messages between two participants that are prepared to send and receive them. In BPMN, two separate Pools in a Diagram will represent the two participants (e.g., business entities or business roles).</p> <p>Compensation Association occurs outside the Normal Flow of the Process and is based upon an event (a Compensation Intermediate Event) that is triggered through the failure of a Transaction or a Compensate Event. The target of the Association must be marked as a Compensation Activity.</p>	
<p><b>Association</b></p>	<p>An Association is used to associate information with Flow Objects. Text and graphical non-Flow Objects can be associated with Flow Objects. An arrowhead on the Association indicates a direction of flow (e.g., data), when appropriate.</p>	

Pool	A Pool represents a Participant in a Process also acts as a “swimlane” and a graphical container for partitioning a set of activities from other Pools, usually in the context of B2B situations.	
Lane	A Lane is a sub-partition within a Pool and will extend the entire length of the Pool, either vertically or horizontally. Lanes are used to organize and categorize activities.	
Data Object	Data Objects are considered Artifacts because they do not have any direct effect on the Sequence Flow or Message Flow of the Process, but they do provide information about what activities require to be performed and/or what they produce.	
Group	A grouping of activities that are within the same category. This type of grouping does not affect the Sequence Flow of the activities within the group. The category name appears on the diagram as the group label. Categories can be used for documentation or analysis purposes. Groups are one way in which categories of objects can be visually displayed on the diagram.	
Text Annotation	Text Annotations are a mechanism for a modeler to provide additional information for the reader of a BPMN Diagram (“Text Annotation” on page 94).	

## 4.2. Intalio|BPM Community Edition

The main tool used in the implementation of the PLETIN method is the open source version of Intalio|BPM Community Edition. Intalio|BPM Community Edition is a complete BPM solution providing two components: Intalio|BPM Community Edition and Intalio|Designer. Intalio|Designer is the process modeling component. It supports BPMN modeling in an Eclipse-based environment.

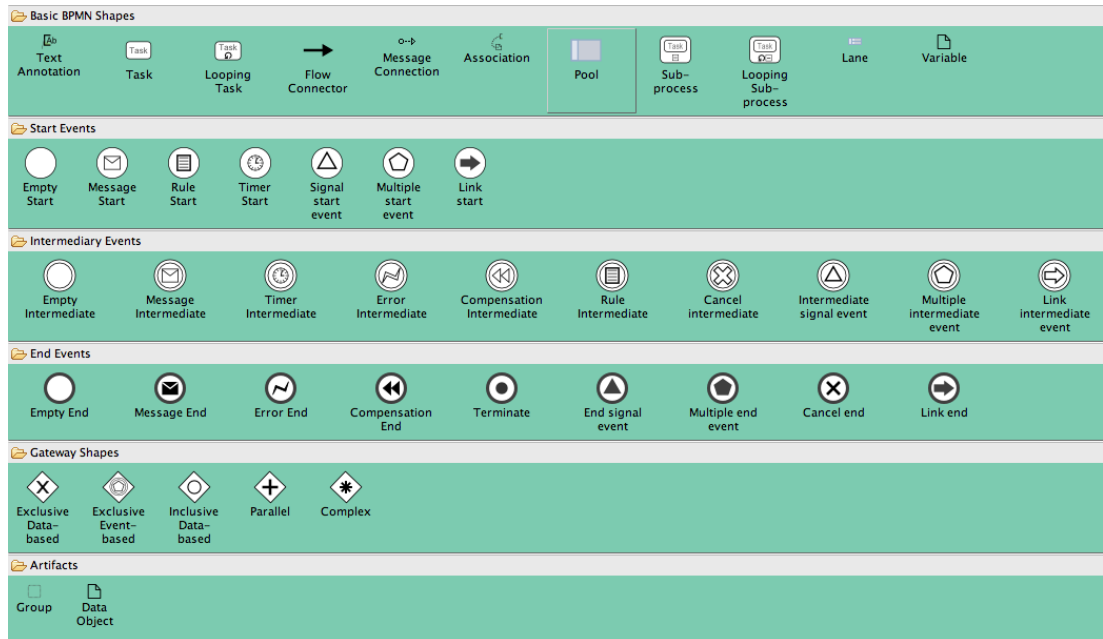


Figure 30 Supported BPMN elements in Intalio|Designer

Intalio|Designer provides the following facilities that is used during the application of the PLETIN method:

1. Development of process models using the BPMN notation. (See Figure 30)

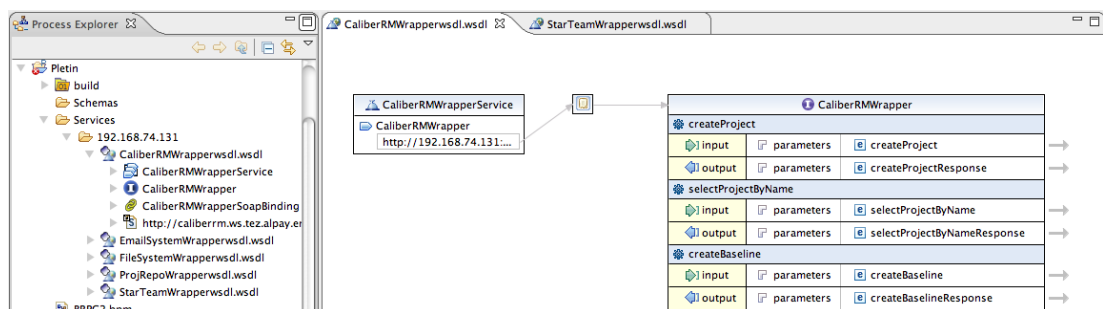
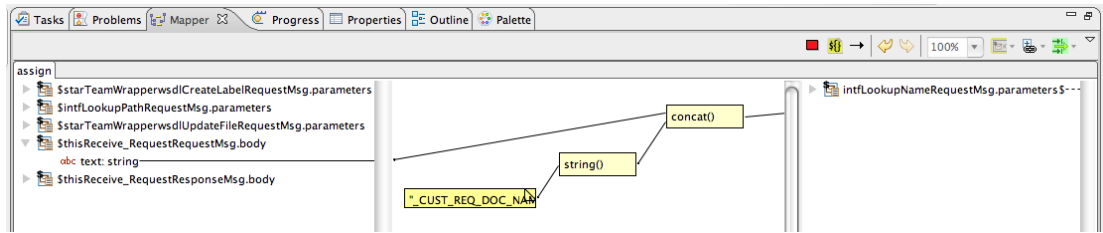


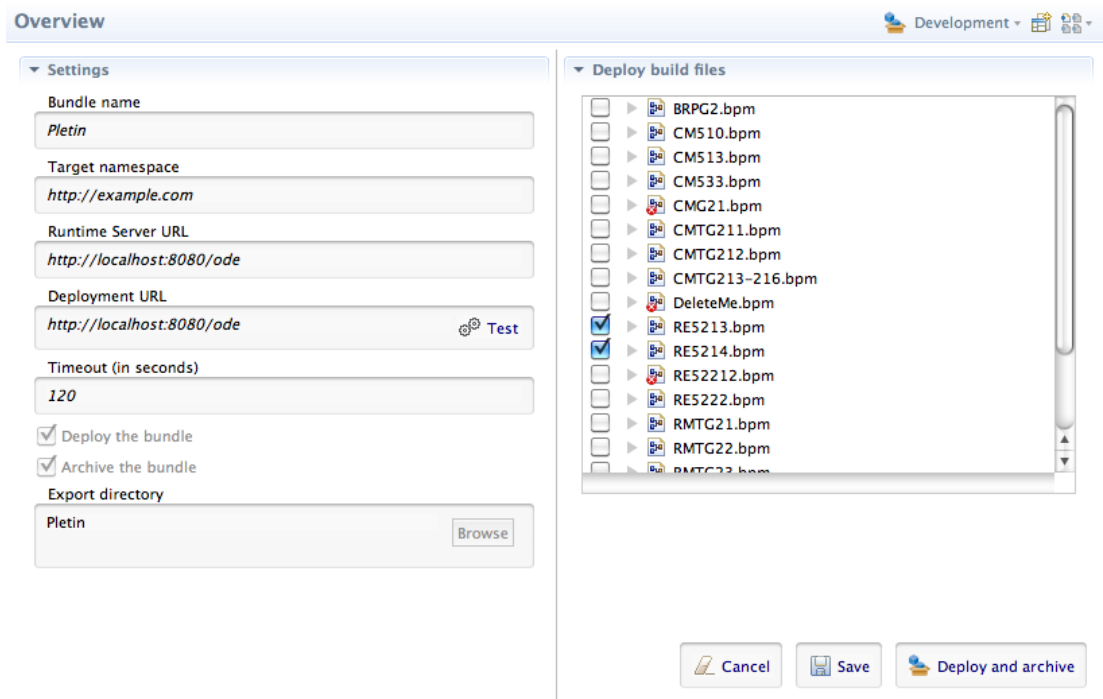
Figure 31 Web-service definitions imported to the workspace

2. Validation of BPMN models, highlighting missing elements, consistency check and ability to execute check.



**Figure 32 Data mapping in Intalio|Designer**

3. Ability to import web-service definitions in the form of WSDL files directly into the designer workspace to embed them into the process models. Processes can invoke and consume these web-services during execution. (See Figure 31)
4. Data mapping to ensure data that is supplied by the user and generated during the execution of the process is correctly routed. (See Figure 32)
5. One-click deployment of processes to Intalio|BPM Community Edition for execution. (See Figure 33)



**Figure 33 Intalio|Designer process deployment dialog**

Process models developed in Intalio|Designer are deployed on the Intalio|Server, where they are converted to BPEL and executed. Intalio|Server is a J2EE [40] based BPM suite which embeds the Apache ODE BPEL engine [4]. It provides the following facilities that, can be employed during the application of the PLETIN method:

1. Web interface where users are only presented with information (including available processes, process diagrams, execution summary for instances) and actions they are permitted to access based on their role (See Figure 34 and Figure 35).

Process	Lifecycle	In Progress	Failure	Suspended	Failed	Terminated	Completed	Total
<input type="checkbox"/> AbsenceRequest [v1]								
<input type="checkbox"/> AbsenceRequest	ACTIVE	-	-	-	-	-	-	-
<input type="checkbox"/> Denemeler [v1]								
<input type="checkbox"/> TimeExample:TimeProcess	ACTIVE	-	-	-	-	-	2	2
<input type="checkbox"/> HelloWorld [v1]								
<input type="checkbox"/> HelloWorld:HelloWorld	ACTIVE	-	-	-	-	-	1	1
<input type="checkbox"/> Pletin [v8]								
<input type="checkbox"/> RE5213:RE5213_Seq6	ACTIVE	-	-	-	-	-	1	1
<input type="checkbox"/> RE5214:RE5214_Seq7	ACTIVE	-	-	-	-	-	-	-
<input type="checkbox"/> TaskManager [v1]								
<input type="checkbox"/> TMP:TaskManagementProcess	ACTIVE	-	-	-	-	-	-	-
<input type="checkbox"/> TezDeneme [v12]								
<input type="checkbox"/> CreateFolder1:Process	ACTIVE	-	-	-	-	-	-	-
<input type="checkbox"/> SimpleLookup1:Process	ACTIVE	-	-	-	-	-	-	-
<input type="checkbox"/> TwoStepLookup:Process	ACTIVE	-	-	-	-	1	-	1
9 processes	9 Active 0 Retired	0	0	0	0	1	4	5

Figure 34 Intalio|BPM Community Edition process operations interface

2. Ability to initiate processes through the web interface.
3. Ability to monitor process execution and access detailed information during and after execution (See Figure 36).

4. Ability to work on different versions of the same process through automatic versioning.
5. Ability to undeploy/retire processes.

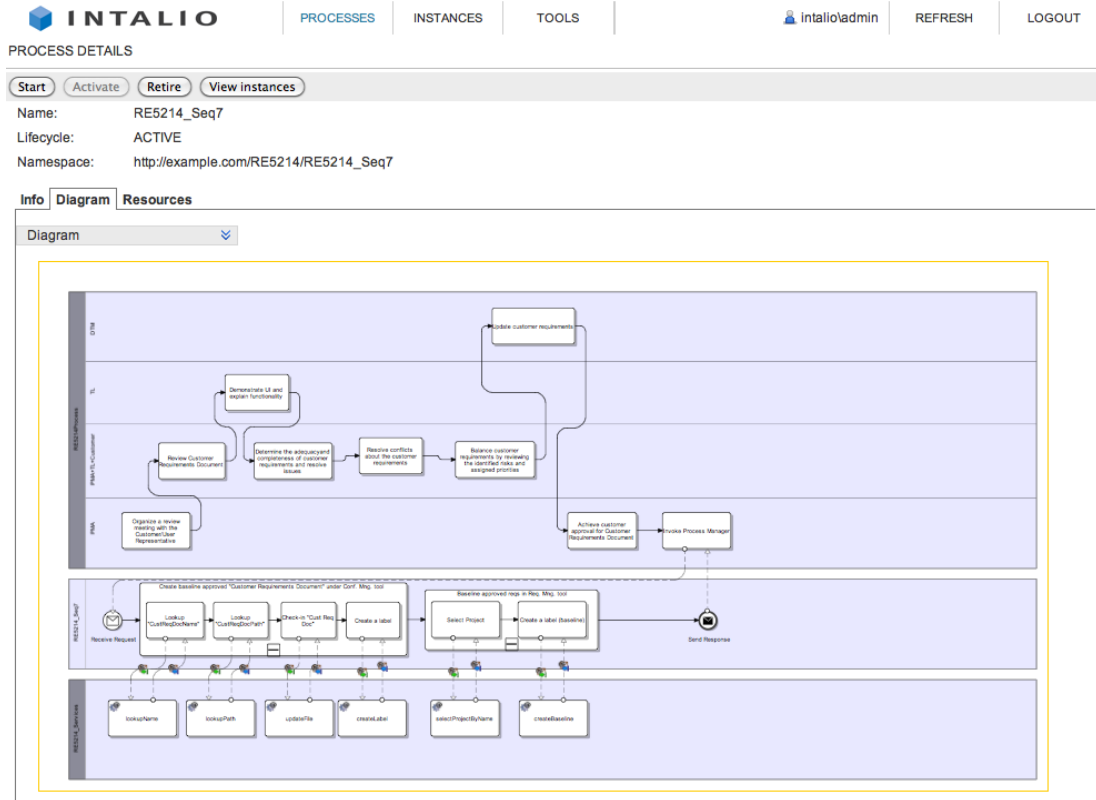


Figure 35 Intalio|BPM Community Edition process detail interface

Using Intalio|BPM Community Edition it is possible to develop custom forms where users enter required information for process execution and access forms available to their roles.

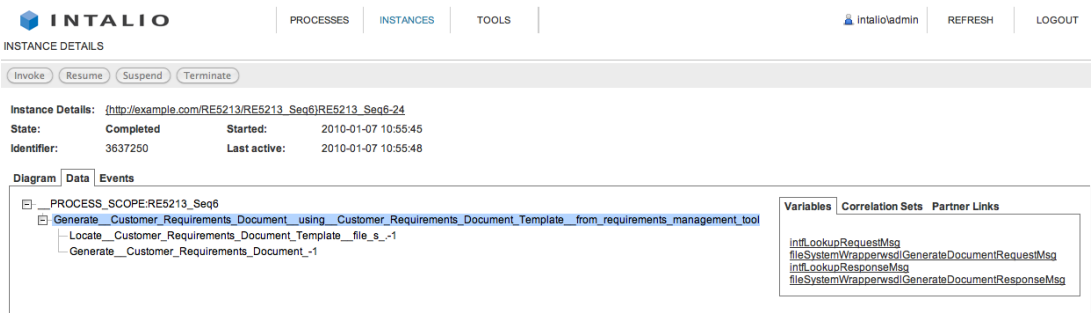


Figure 36 Intalio|BPM Community Edition process instance detail interface

### 4.3. Eclipse and Apache Tomcat

Web-services used for integration implementations were developed using Java language on Eclipse [24], an open source development platform. Web-services were generated directly from standard Java classes using the “web-service wizard” (See Figure 37) provided by Eclipse and deployed to Apache Tomcat [5], an open source software implementation of the Java Servlet [41] and JavaServer Pages [42] technologies. API documentation is used to understand the APIs provided by software development tools.

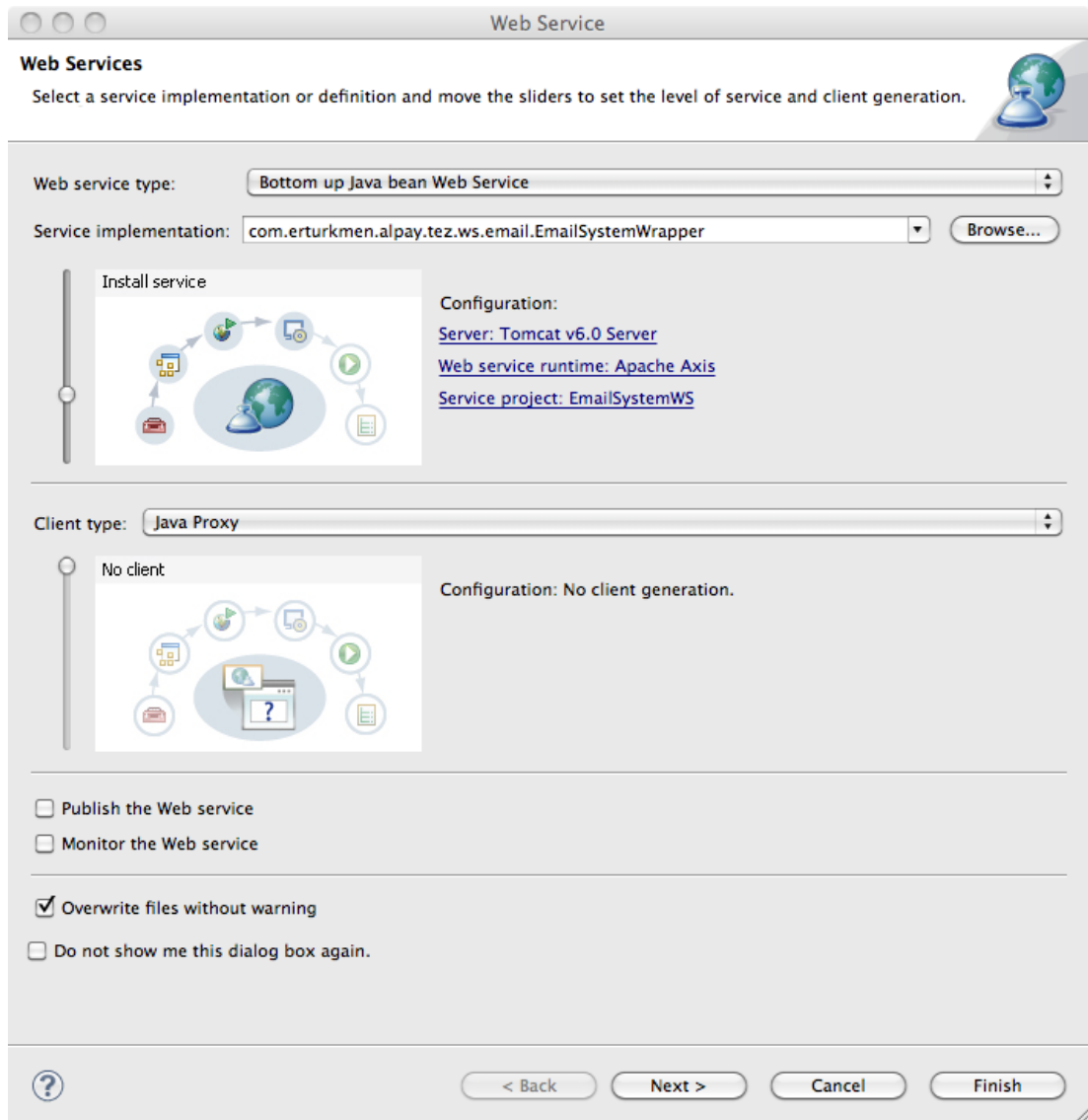


Figure 37 Eclipse "web-service wizard"

## **CHAPTER 5**

### **CASE STUDIES**

To develop a method that provides a solution for tool integration, we have performed two case studies involving two target organizations. In the first case study (Case Study I) we have focused on user interactions with tools to understand the extent of tool use and then studied the issues with tool integration to verify their existence and significance. Based on these findings we have developed the PLETIN method. On the final phase of Case Study I we studied the applicability and efficiency of the PLETIN method through a prototype implementation.

Case Study I provided insights into the issues with tool integration in software development. A second case study (Case Study II) was designed and performed focusing only on the conduct of PLETIN. The aim of Case Study II was to understand the applicability of the PLETIN method on some other environment than Organization I. A case study in a separate environment was required since the method was developed based on the results of Case Study I and was dependant on them.

The PLETIN method is based on the scenario where there are no integrations between the tools in a software development environment; cooperation of them must be maintained manually. As an example, to make it possible for different tools that



are not integrated to work on the same data set, the data must be fed to each tool manually. Similarly, for a tool to operate on the information created by another tool, data should be moved between the tools by a user manually. In other words, users must perform actions necessary to keep the tools working together (cooperate). In this situation, process definitions (or models derived from these definitions if they exist) would contain sequences of actions (what we name integration-tuples or sequences, and use interchangeably in this thesis) performed to maintain tool integrations. These sequences are required to keep the tools working cooperatively. Thus, user actions account for non-existing integration facilities of the tools. For simplicity, we call these facilities integration-tuples (or sequences). An integration-tuple is a candidate tool integration situation. In our scenario users maintain integration-tuples manually. A method designed to investigate the process models can be used to understand how users interact with the tools to maintain these tuples. Based on this knowledge, requirements for the services to support these actions can be inferred and implemented to build a tool integration framework.

We have selected a multiple case study design as our research method since our research presents ‘how’ and ‘why’ questions to understand the extent of tool use and observe problems with tool integration. The behavioral nature of the problems we are dealing with, and the difficulties of observing results in an experimental setting prevents us from trying other methods where we are required to modify the behavior we are investigating.

Case Study I was performed at Organization I, which is a software development organization that has process maturity certified as CMMI Level 3. The organization had clearly-defined process definitions using a multitude of tools for software development.

Case Study II was performed at Organization II, which is a software/systems development organization that has process maturity certified as CMMI Level 3. Organization II encouraged use of tools in their software development processes, however the use of tools is not mandatory.

Both organizations significantly contributed to this research by providing access to their process definitions for analysis while answering our questions about tool use and integration during the case studies.

This chapter is organized as follows: Section 5.1 describes how the multiple case study approach was designed, including questions for individual case studies. Section 5.2 defines constraints on the case selection. Section 5.3 and Section 5.5 detail the execution of the two case studies along with samples of artifacts produced. Section 5.4 and 5.6 present the results of the case studies. Section 5.7 is on the validity threats for the case studies. Section 5.8 summarizes the results of the case studies and provides a discussion on the results.

## **5.1. Multiple Case Study Design**

A multiple case study design was used in this thesis. Two case studies were planned to develop and then validate the applicability and efficiency of the PLETIN method. The first case study was designed as an exploratory case study to observe and validate the existence of tool integration issues in software development. Based on these issues a method for tool integration was developed. The PLETIN method was applied in two different organizations to observe its applicability and efficiency.

### **5.1.1. Case Study I Design**

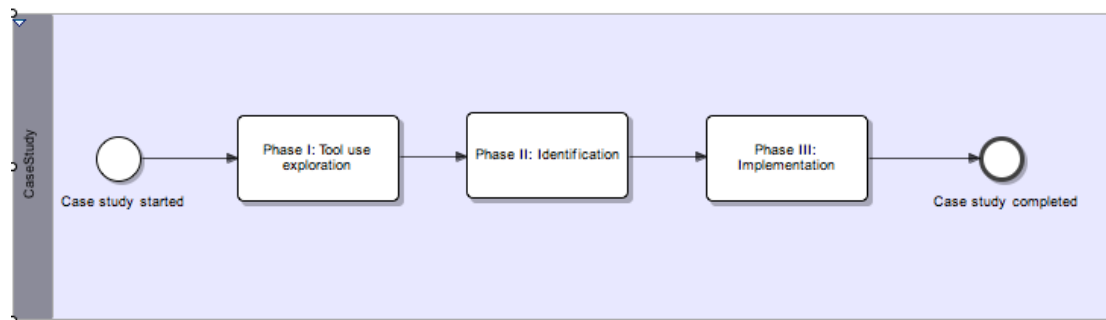
PLETIN is a process modeling method developed to provide a solution to tool integration problem. PLETIN is used to identify tool integration-tuples required by an organization and implement them. The main research question for the case study was defined as:

"What is the applicability and effectiveness of the PLETIN method in identifying and implementing tool integration-tuples in software development?"

An integration-tuple is defined as an ordered list of independent tools or different parts of the same tool providing separate features and an action. It can be represented as "(ToolA, ToolB...ToolN, ActionX)" where ToolA, ToolB, and ToolN stand for tools and ActionX denotes the sub-process requiring these tools. The tuple

corresponds to an integration situation between the tools used to perform “ActionX”. A tuple can be defined both for an existing implementation or only denote a possibility of such an integration. Integration-tuples represent a series of simple (one of Create, Read, Update, Delete and Execute operations) user interactions with tools, thus they are interchangeably called sequences in this thesis.

The case study was designed to have three phases (See Figure 38). The first phase was the tool use exploration phase. Its aim was to provide a basis for the other two phases that focus on the development, implementation and evaluation of the method.



**Figure 38 Process model for Case Study I Design**

The identification phase aimed to unravel the extent of user interactions with tools [26]. In this phase the goal was to understand if the tools and the tool integration problem have an effect in practice. The research question for the first phase was:

"How extensively are tools used in software development?"

To answer this research question, software development process definitions were analyzed to identify user interactions with tools. Process definitions not containing any tool interaction were excluded at the beginning of the case study to reduce the scope, since by definition they can't contain any tool interactions. Interactions in all processes were identified to understand the extent of tool use, coupled with the ratio of processes having interactions and the distribution of interactions in process areas.

The extent of tool interactions observed in the first phase indicated a strong dependency on tools from users and software development processes. Because of this dependence, existing features, strengths and limitations of the tools directly impact the user and process performance. While an enhancement in tool features would

increase the process performance, a reduction would result in poor-performing processes [44].

[45] suggests that an integrated toolset would give better results, benefitting the execution of software development processes. The second phase of the case study (the identification phase) focused on the existence and significance of the tool integration problem to understand the status of tool integration in the organization, to observe problems related to it in practice and interpret to what extent the PLETIN method would be helpful [27]. The research question for this part was:

"What is the significance of tool integration problem in software development?"

[45] states that for a tool to be used effectively in an information systems lifecycle process it must fulfill "the specific needs and expectations of the organisation, and its associated stakeholders". To understand the state of tool integration, the identification phase focused on what kind of integration features the users need in order to execute their processes. These integration features required by the users (and processes) were denoted by integration-tuples.

Process models based on process definitions conforming to certain criteria were defined to identify these tuples. These models were used to identify sequences of simple user interactions with tools, i.e. integration-tuples. This set of integration-tuples established the requirements for tool integration based on organizational processes. A gap analysis comparing these requirements to the existing state of tool integration in the organization was executed. The frequency and the process area distribution of these tuples were also noted. The results described the nature and significance of the tool integration problem and guided our efforts on developing the PLETIN method to provide a solution.

We have developed the PLETIN method to identify and implement integration-tuples. Integration-tuples were identified at the end of the second phase. The third phase of the case study was conducted to observe the applicability and efficiency of the method in implementing integration-tuples [27]. The sequences identified in the process models were broken down into atomic actions. Corresponding services (or

interfaces), which the organizational processes require from the tools, were defined. The definitions of the services were built from input and output messages exchanged and the action normally carried out manually by a user. Service definitions in the form of WSDL files were combined into process models. This enabled the implementation of integration-tuples as business processes executed by a business process management suite. Business services were built such that they can consume the implemented services on behalf of users. Data for the effort spent on the implementation was recorded. Also, information on manual execution of the processes was obtained and compared to the automated execution case.

The results of this case study show that software development processes are dependent on the use of tools. The integration between tools are however not satisfactory for the processes because many opportunities were missed. These missed opportunities were significant and extended to all stages of software development. The frequency of use ranged widely from once per requirement revision to once per project. The results suggests that it is feasible to identify these opportunities and provide custom implementations for them using the PLETIN method implemented in the second and third phases of the case study.

### **5.1.2. Case Study II Design**

To understand the applicability of PLETIN method on a separate but similar environment the following questions were developed and answers were sought by the application of the PLETIN method:

1. “Is it possible to identify candidate tool integration situations from process definitions using PLETIN?”
2. “Are there any similarities between the service definitions and business processes developed from the two cases?”

The first question is directly concerned with the feasibility and the ease of application of the PLETIN method. It is crucial that the method developed for this thesis is easy to use and provides useful results in an effective manner. An easy to use and efficient method is a must for the adoption of the method. Although the ease

of use and practical results provided by the PLETIN method was observed in Case Study I, an independent case study was required for unbiased results.

The second question aims for a comparison between the results of Case Study I, and the newly planned case study. Users generally interact with tools in similar ways even for performing different processes of different organizations. Radical departures from the common interaction methods can also be observed, however rarely, because the goals of software development are similar. Since the method is used to identify user interactions with tools, it is expected that organizations produce similar results with occasional differences between them. The second question aims to understand and discuss the level of similarities and differences between two case studies.

## **5.2. Case Selection**

The problem under analysis puts the following constraints on the target organization for proper conduct of the case study:

1. The existence and active use of multiple tools for software development
2. A defined set of software development processes (preferably certified in CMMI Maturity Level 3 [18], or comparative ISO 15504 [39] level)

Since this case study is part of a research on the tool integration problem in software development organizations, the setting for conduct of the case study must have a multitude of tools supporting the processes in use.

The case study requires the analysis of process definitions to extract user interactions with tools. Analysis of process definitions was favored over the observation of actual user interactions with tools so the method can be used in parallel with process modeling and/or improvement efforts, while the method of observation do not affect the results.

## **5.3. Execution of Case Study I**

The case study was performed in a software development branch of a research organization. This branch develops software for military and civilian systems. To support their development efforts, they utilize multiple tools. These tools include

requirements management, configuration management, change management, test management, automated functional testing, project planning, risk management and time tracking. The processes for software development were already defined and the organization had recently been evaluated to be CMMI Level 3. We have been able to work on the process definitions and members of the Software Engineering Process Group (SEPG) provided answers to our questions whenever we requested.

### 5.3.1. Tool Use Exploration Phase

In the beginning of the case study, a meeting was held with the department head and the SEPG leader to develop the schedule for the case study work and the SEPG Q&A sessions. After setting up the schedule and sessions, we started the scoping of the case study. Process definitions had been grouped by the organization using a categorization similar to the process area definitions in the CMMI model [21]. Working with the SEPG, we identified the process areas that are not directly related to software development thus are irrelevant to our case study. These process areas focused on process and project management. A later analysis revealed that these process areas have none or single tool interactions (per process definition) thus were not suitable for our research. The process areas considered not relevant to our case study are given in Table 5:

**Table 5 Processes areas not directly related to software development**

<b>Process Name</b>	<b>Process Abbr.</b>
Process Management	PcM
Measurement and Analysis	MA
Software Quality Assurance	SQA
Risk Management	RkM
Organizational Training	OT
Project Management	PM
Decision Analysis and Resolution	DAR

The remaining process areas that were related to software development defined the scope of the case study. The process areas included in the scope of the case study are given in Table 6.

85 process definitions constituting the 4 process areas (CM, RE, TS, VV) were analyzed. User interactions with tools in all processes were identified to understand the extent of tool use, coupled with the ratio of processes having interactions and the distribution of interactions in process areas.

**Table 6 Process areas included in the scope of Case Study I**

<b>Process Name</b>	<b>Process Abbr.</b>
Configuration Management	CM
Requirements Engineering	RE
Technical Solution	TS
Verification and Validation	VV

### **5.3.2. Identification Phase**

The extent of tool interactions observed in the first phase indicated that users and software development processes for the organization had a strong dependency on tools. 90% of all analyzed process definitions contained tool interactions.

The aim of the identification phase was to understand if issues with tool integration were significant for the organization. If it is, then the benefits of the toolset to software development can be increased through better integration. For this we started gathering information on the state of tool integration in the organization, problems related to it in practice and their extent. Thus, the identification phase focused on the existence and significance of the issues with tool integration while unraveling what kind of integration features users need in order to execute their processes. These integration features required by the users (and processes) were denoted by integration-tuples.

To identify these tuples process definitions that included multiple interactions with tools were selected. Processes including complex flows for decision-making, review,



design, creative development and collaboration were excluded because they didn't provide any data on tool integration but rather focused on manual tasks only people could perform. The outcomes of such processes can vary, i.e. they are not deterministic [9]. The remaining process definitions were converted to process models using the BPMN notation. A sample BPMN model developed in this phase is given in Figure 39.

The models were used to visually identify sequences of simple user interactions with tools, i.e. integration-tuples. To identify integration-tuples, process models developed from process definitions were used. These process models represent tool interactions by a BPMN message connection between a task performed by a user and a BPMN pool representing a tool. Tasks containing tool interactions were classified depending on whether they were complex or not. An action was classified as simple if it was one of the CRUD operations: Create, Read, Update, Delete or Execute. Simple tool interactions were highlighted with a distinctive color (e.g. orange) on the process model. A sample for the representation of tool integrations in a process model is given in Figure 40.

The highlighted tool interactions were grouped together using BPMN group objects to represent a sequence. A sample of such grouping is given in Figure 41. These sequences, consisting of multiple, simple tool integrations present tool integration opportunities. See a detailed discussion in 3.4.3. These sequences of actions are labeled as integration-tuples.

Integration-tuples were then organized into an "integration map" providing a graphical representation of the organizational process requirements for integration. This integration map was compared to the "current integration map" representing all available integration implementations for the tool set, whether they were used by the organization or not.

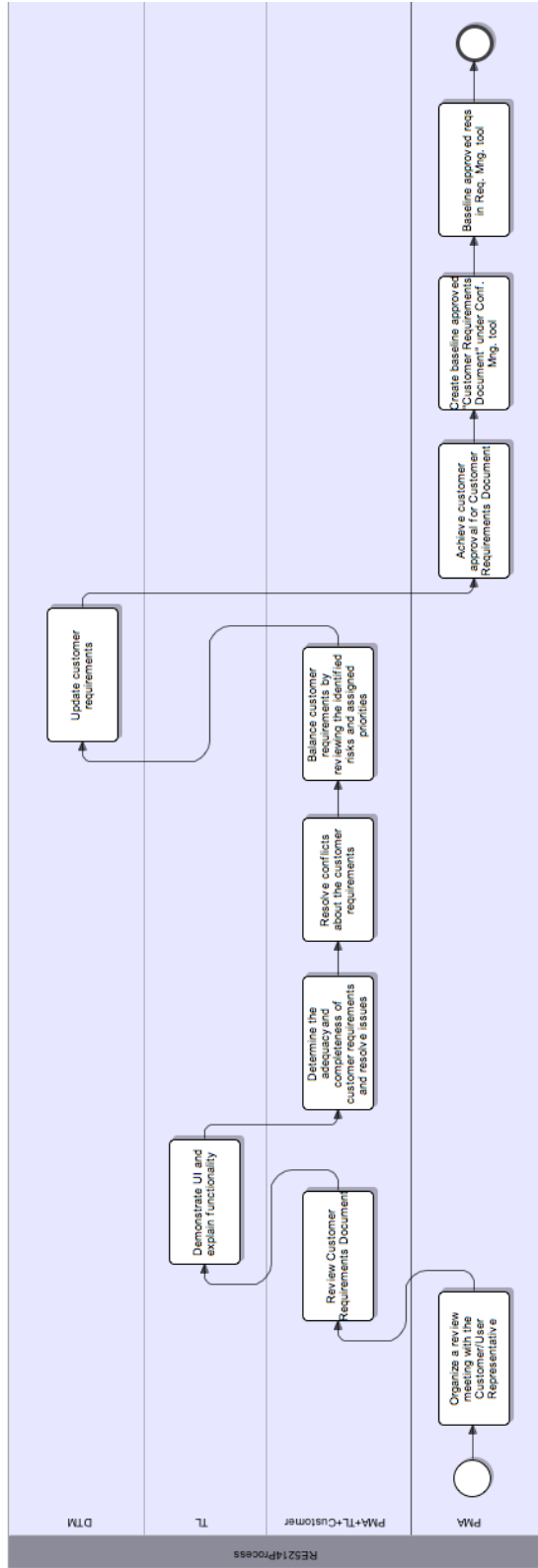
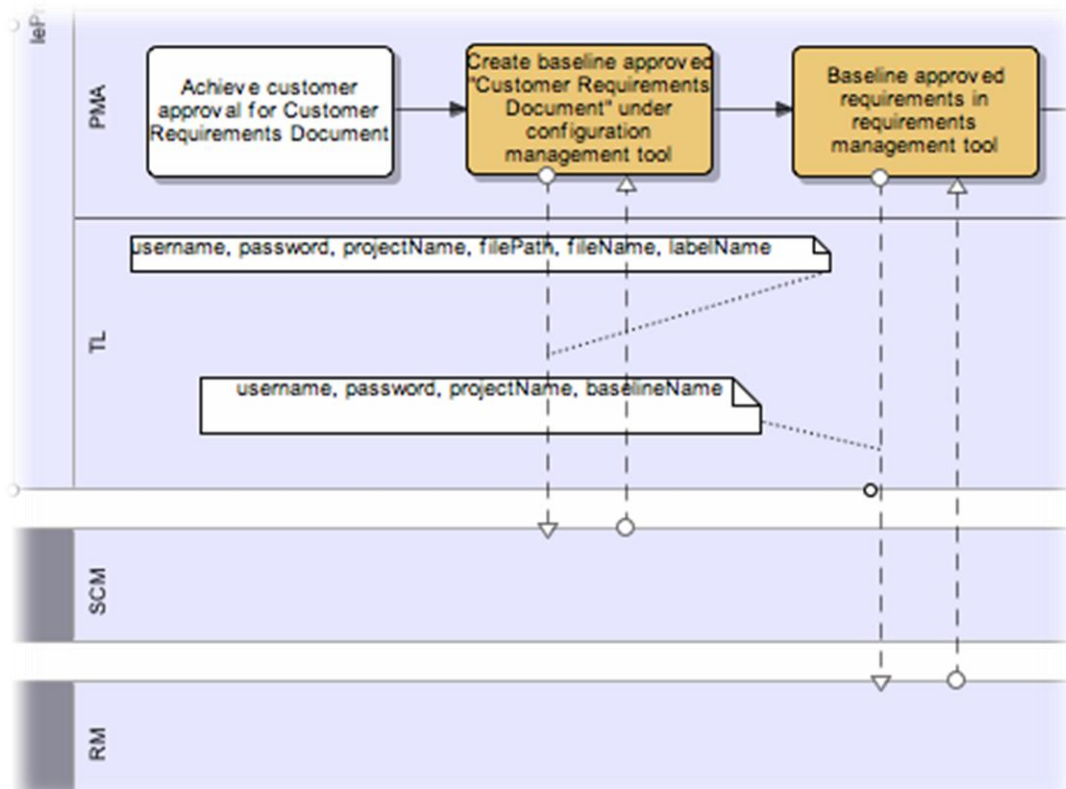


Figure 39 Sample BPMN model for Case Study I, Phase I, RE5214

A gap analysis was performed comparing the two maps. This analysis comparing the requirements to the existing state of tool integration clearly laid out the missing integration implementations for the organization and the significance of the tool integration problem experienced by the users. We also noted the frequency and the process area distribution of these tuples. The results described the nature and significance of the tool integration problem and guided our efforts on developing the PLETIN method to provide a solution.



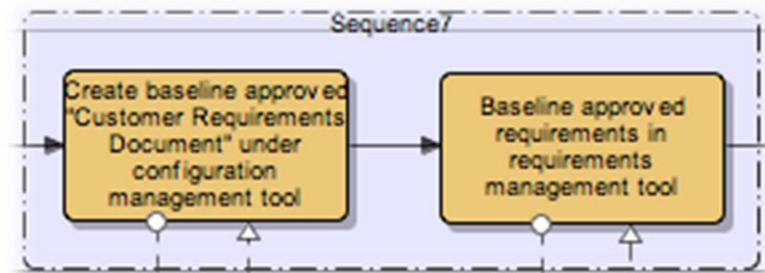
**Figure 40 Sample tool interaction represented as a process model**

### 5.3.3. Implementation Phase

The next step after the identification of unsatisfied integration-tuples was the implementation step. Since the process models were already developed in the previous phase, the implementation phase was concerned with the identification of the implementation details.

The integration-tuples defined in the previous phase were used as the requirements or high-level definitions for the services to be developed. The implementation effort

used top-down and bottom-up approaches for Service Oriented Application method [7]. In this approach, both existing services and the requirements are reviewed, modified and used to reach an acceptable solution. The integration-tuples (sequences) and the actions constituting them were broken down into atomic actions. The atomic actions were compared with existing services provided by the tools. If there was a match, the service was used as the implementation of the action.



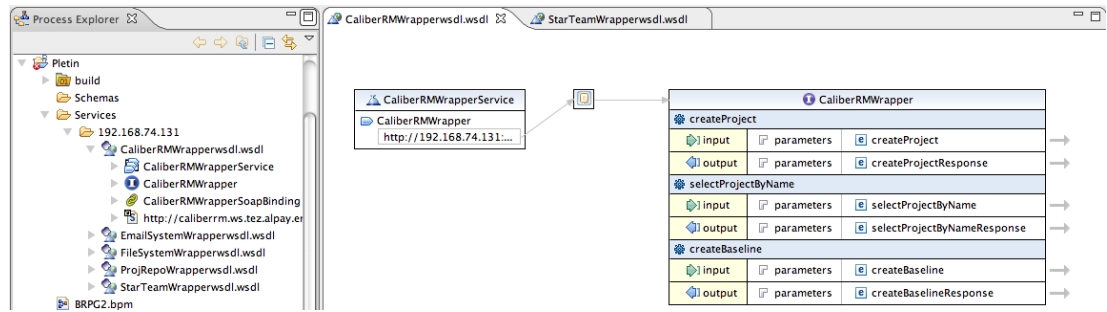
**Figure 41 Two actions grouped into a sequence**

In this thesis, web-services technology was used since they are standard-based. If the tool did not provide a web-service implementation, a wrapper was developed. If the specification of the action corresponded to several web-services, the web-services could be combined into a new service, or the action definition could be further broken down. The action definition could be modified to use already existing services rather than implementing a new one, if the action was very similar to the web-service. However, if there was no corresponding or similar existing web-service for an action, then custom implementation was necessary.

In this case study, we have implemented the required web-services ourselves. In other cases software developers in the organization can assist the process group, or even vendor assistance can be sought for. Custom implementations were developed complying with the requirements based on the atomic action, and input/output messages depicted on the process model.

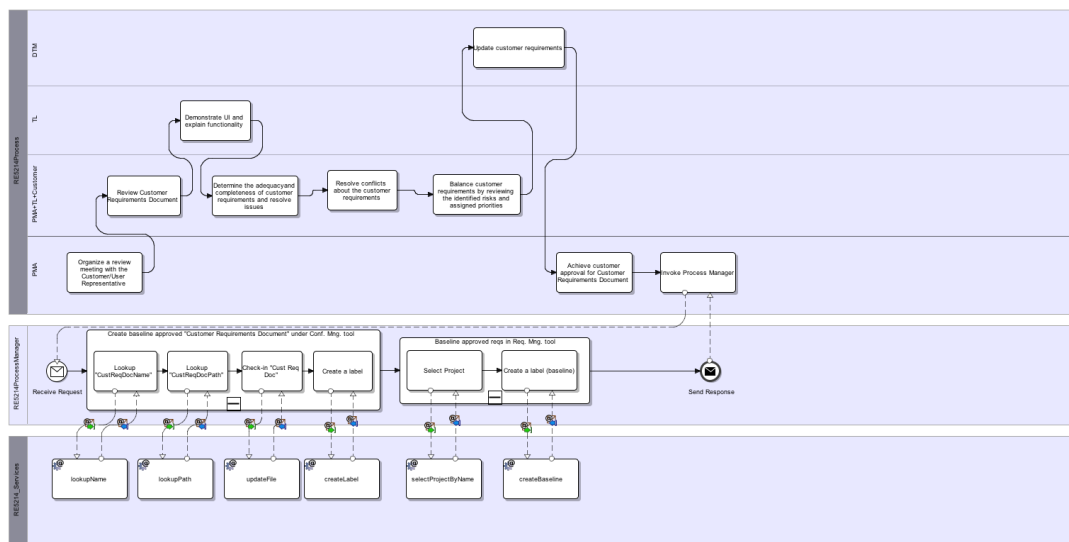
When all atomic actions were mapped to an existing service and required web-service implementations were completed, they were imported into the BPMN modeling workspace (See Figure 42). The web-services were deployed into a separate BPMN pool labeled as “Tools”. Message connections were created between

the atomic actions and web-services. Atomic actions connected to web-services were moved to a separate BPMN pool labeled as “Process Manager” to accommodate the processes to be executed by the integration framework on behalf of users. This way, all actions normally performed manually by users were delegated to the integration framework. Users could experience an integrated toolset since the framework performs actions otherwise manually executed by them.



**Figure 42 Web service definitions imported to the workspace**

After the process models were completed and reviewed, they were deployed to the Intalio|BPM Community Edition for review, testing and execution. A sample for the process models completed and deployed for execution is given in Figure 43. For all process models developed in the case study, see APPENDIX D: PROCESS MAPPING (CASE STUDY II).



**Figure 43 Sample completed BPMN model for Case Study I, RE5214**

### 5.3.4. Discussion on Implicit Sequences for Case Study I

UML modeling tool is mentioned as utilized by the development team in TS process area. Details on how the tool should be used are available in process definitions. However, sequences of tool interactions with other tools are not observed and tool appears to be used as standalone. This required further analysis for implicit sequences in Case Study I and revealed two sequences.

The first one is classified as an omitted/unmentioned implicit sequence where document generation action from the tool is not explicitly stated but exists as a tool feature. Process model for TS512 is given in Figure 44, where a tool interaction (generate document) is omitted and defined as a manual action. Tool features are used to identify this interaction and sequence it makes up.

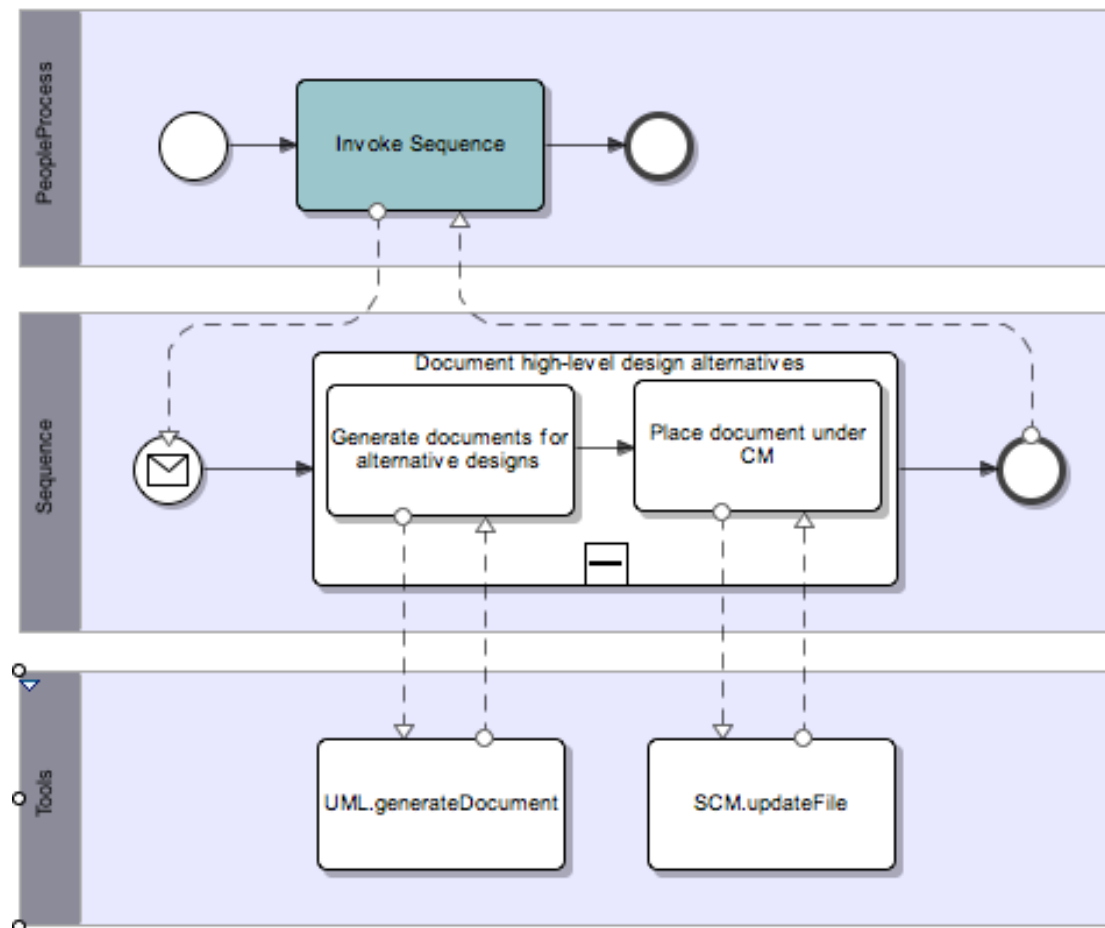
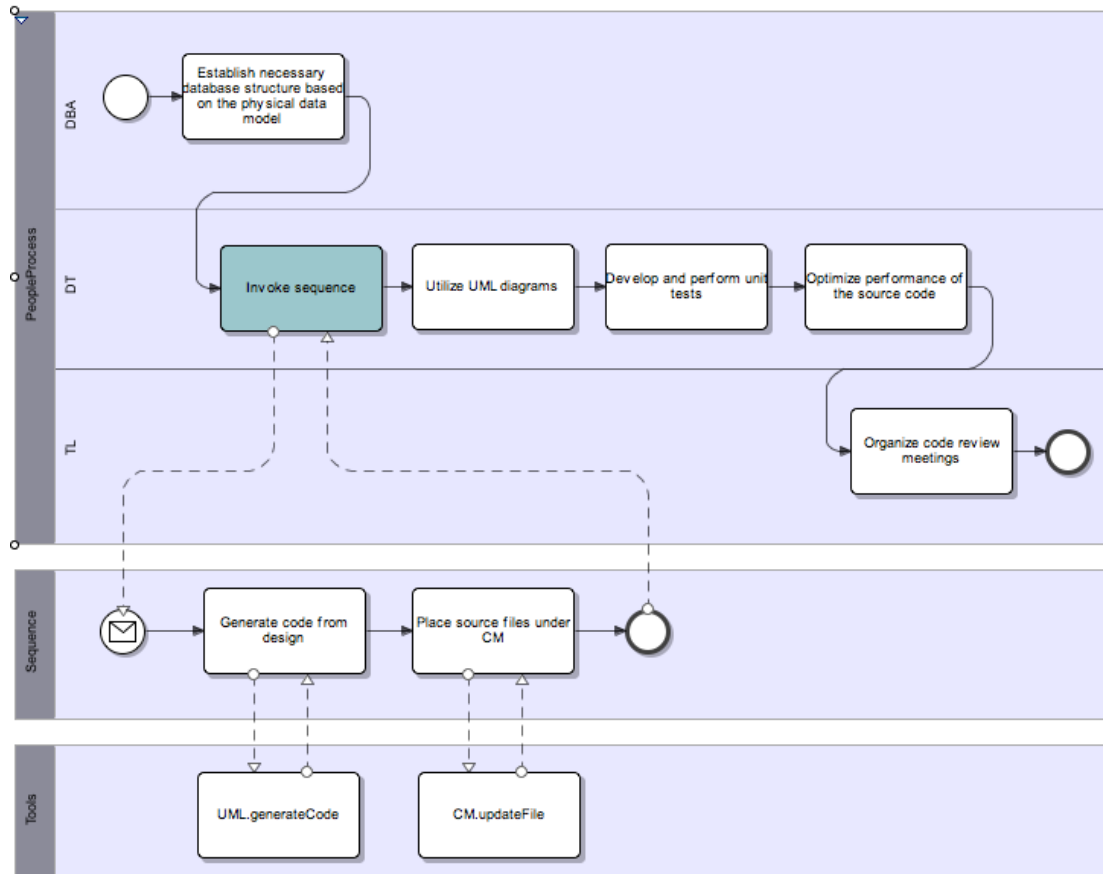


Figure 44 Process model for TS512

Another implicit sequence was revealed in process TS521. It was classified as an interrupted sequence because the code generation from the UML/IDE tool does not directly precede submission to configuration management tool in process definitions. The submit action was further analyzed to uncover this implicit sequence. Process model for this sequence is given in Figure 45.



**Figure 45 Process model for TS521**

Process definitions analyzed in CS1 does not include any mentions of IDEs except TS521. However analysis of the tool features reveals existing integrations with other tools like the configuration and change management tools. Unfortunately, the integration of IDEs with other tools like SCM is specified in process definition documents. This hints several possible unmentioned/omitted and compound implicit sequences in process definitions. Improvement of these process definitions through observation and process discovery is left as future work.

#### 5.4. Results for Case Study I

The tool use exploration phase of the case study analyzed the process definitions of the target organization. There were a total of 85 process definitions from the four process areas we have investigated. 77 of these process definitions contained interactions with tools. The distribution of these interactions with respect to process areas is given in Table 7.

43 of 85 process definitions (51%) we have analyzed are labeled to be completely creative (unstructured) processes, including review, approval, analysis, design and development activities. These processes were considered not suitable for tool integration.

30 of the 85 process definitions we have analyzed for the case study contained candidate integration situations. The distribution of these candidates with respect to the process areas is given in Table 8.

**Table 7 Tool interactions with respect to process areas**

<b>Process Area</b>	<b># of proc.def.</b>	<b>With tool interaction</b>
<b>CM</b>	17	11
<b>RE</b>	11	11
<b>TS</b>	13	13
<b>VV</b>	18	17
<b>Docs/Guidelines</b>	26	25
<b>Total</b>	85	77 (90%)

It is notable that, the majority of the candidates were derived from supporting documents and guidelines for tool use. This was expected since these documents describe how the tools should be used and consider the features available from the tools to a greater extent than the other definitions.



**Table 8 Distribution of candidate integration sequences with respect to process areas**

<b>Process Area</b>	<b># of proc. def.</b>	<b># of candidates</b>
<b>CM</b>	17	5
<b>RE</b>	11	4
<b>TS</b>	13	2
<b>VV</b>	18	1
<b>Docs/Guidelines</b>	26	18 (60%)
<b>Total</b>	85	30 (35.3 %)

Further analysis of these candidates revealed references and overlaps between process definitions. For duplicate process definitions, those with the highest detail were chosen. Process definitions in guidelines consistently had more detail. Discarding these duplicates, a total of 26 integration-tuples were identified. Almost half (42%) of all tuples were from guideline documents. Remaining tuples were uniformly distributed to other process areas. The distribution of these tuples to process areas is given in Table 9. The distribution of the execution frequency for the tuples is given in Table 10.

**Table 9 Distribution of tuples with respect to process areas**

<b>Process Area</b>	<b># of proc. def.</b>	<b># of tuples</b>
<b>CM</b>	17	5
<b>RE</b>	11	6
<b>TS</b>	13	2
<b>VV</b>	18	2
<b>Guidelines</b>	26	11 (42%)
<b>Total</b>	85	26

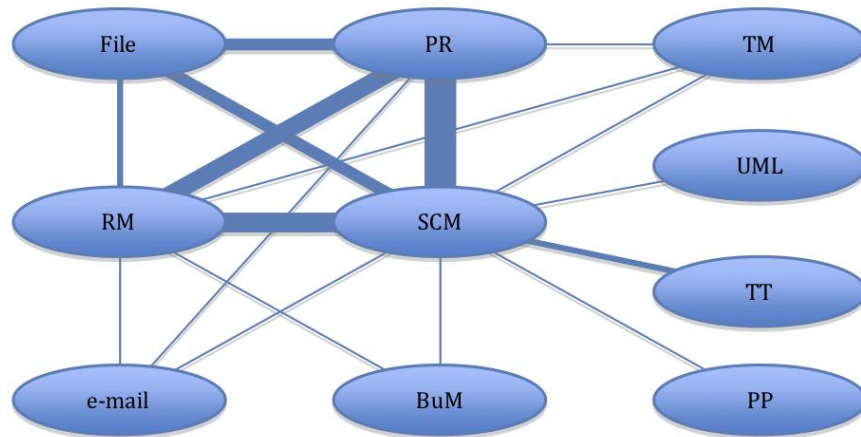
The majority of integration-tuples are executed once per project. This is due to the clear description of how a project is set up and closed in the process definitions. Following per project executions; per SRS release, per change request (CR) or requirement and per build or release executions are observed. The execution frequency of tuples ranges from several times a day to once per project duration (usually around 12-24 months). However, the highest number of executions for a tuple is per CR or requirement since they are executed several times a day.

An integration map was assembled from these integration-tuples. It is given in Figure 46. The integration map visually represents the tuples for each tool to provide an understanding of the requirements of process definitions. The thickness of the connections between tools represents the number of tuples between. The actual number of tuples constituting the integration map is given in Figure 48.

**Table 10 Execution frequency of tuple**

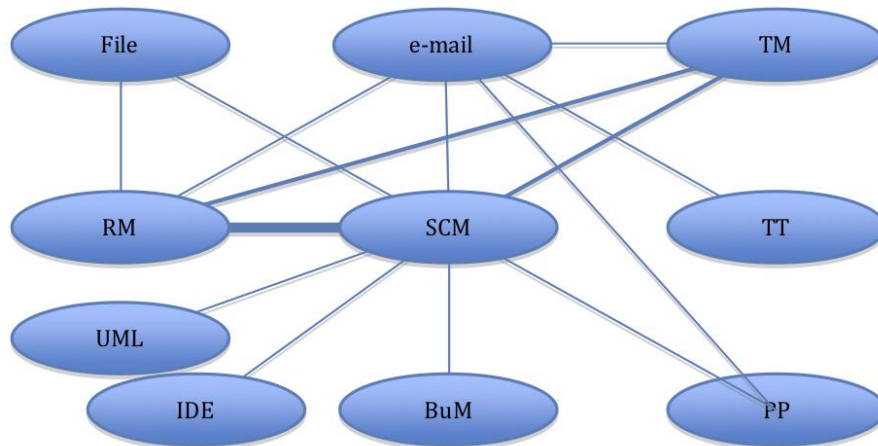
<b>Frequency</b>	<b># of tuples</b>	<b>Percentage (%)</b>
<b>Per project</b>	9	36
<b>Per SRS release</b>	8	28
<b>Per build or release</b>	4	16
<b>Per CR or requirement</b>	5	20
<b>Total</b>	26	100

As it can be seen from the map, 26 tuples use the following 10 systems: software configuration management tool (SCM) Borland StarTeam, requirements management tool (RM) Borland CaliberRM, project repository (PR), UML Modeling Tool (UML) Enterprise Architect, test management tool (TM) HP Mercury Quality Center, project planning tool (PP), time tracking tool (TT), build management tool (BuM), file system (File) and e-mail (e-mail) system. The most significant integration requirements include File, PR, RM and SCM systems.



**Figure 46 Integration map for the case study**

Figure 47 depicts a similar map, built from the existing integration-tuples already in use by the organization. Comparing the two maps, it is clear that the organization is employing a small number of integration-tuples. All of these tuples are point-to-point, and are supplied by vendors for specific versions of the tools in use. There are no custom or 3<sup>rd</sup> party integration-tuples in use.



**Figure 47 Existing integration map of the organization**

It should be noted that, project repository is a simple database storing project information like the path to document templates, or login information. It is developed for the purpose of integration during this case study. It consists of single key-value pairs in a database table and a web-service responding to request including “keys”.

The organization we have performed our case study uses Borland StarTeam [12], a software configuration management tool providing both configuration management and change management features in a single package. For this case study we have not separated its functionalities into two logical tools but rather adhered to the existing features and labeled the tool as software configuration management (SCM) tool.

The modeling effort produced 18 process diagrams. These diagrams are available in APPENDIX D: PROCESS MAPPING (CASE STUDY II). It took 20 hours for a single researcher to complete the modeling effort.

26 integration-tuples identified in the identification phase of the case study were further examined in the third phase. From these 26 integration-tuples, a total of 232 operation calls were identified. Further examination of these calls revealed a need to merge several calls (mostly login and context setting calls), and add new calls (lookup calls from the project repository for context identification).

	BuM	e-mail	File	PP	PR	RM	SCM	TM	TT
BuM	-	0	0	0	0	1	1	0	0
e-mail	0	-	0	0	1	1	1	0	0
File	0	0	-	0	6	3	6	0	0
PP	0	0	0	-	0	0	1	0	1
PR	0	1	6	0	-	9	16	1	0
RM	1	1	3	0	9	-	10	1	0
SCM	1	1	6	1	16	10	-	1	3
TM	0	0	0	0	1	1	1	-	0
TT	0	0	0	1	0	0	3	0	-

**Figure 48** Number of tuples constituting the integration map

The final 145 individual calls constitute a set of 49 unique operations. The distribution of these operations with respect to individual tools is given in Table 11.

The complete list of operations is listed in the APPENDIX A: COMPLETE LIST OF OPERATIONS DERIVED FROM PROCESS MODELS (CASE STUDY I). Services for SCM, RM, PR, file system and e-mail system were implemented. Remaining implementations are planned and left as future work. Definitions for the web-services developed are available in APPENDIX G: DEFINITIONS FOR WEB SERVICES (CASE STUDY I), along with the actual implementations in APPENDIX H: APPLICATION CODE DEVELOPED FOR WEB SERVICES (CASE STUDY I). This corresponds to the implementation of 18 out of 26 (69%) tuples with 40 out of 49 (81%) operations. This effort took a total time of 80 hours for a single researcher who is familiar with the tools but is not an experienced developer.

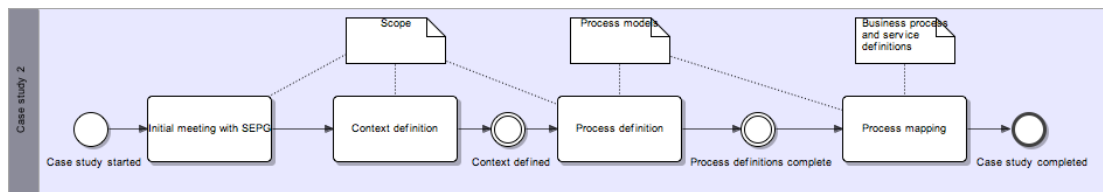
**Table 11 Distribution of operations to tools**

<b>Individual tools</b>	<b># of operations</b>
<b>SCM</b>	22
<b>RM</b>	11
<b>File</b>	5
<b>PR</b>	1
<b>UML</b>	1
<b>BuM</b>	1
<b>TM</b>	3
<b>TT</b>	2
<b>PP</b>	2
<b>E-mail</b>	1
<b>Total</b>	49

## 5.5. Execution of Case Study II

The selection for the second case was based on the constraints on the first case study. A software and systems development organization employing multiple software development tools having already existing process definitions was sought. Organization II was chosen, which is specialized in military systems and have processes that are assessed as CMMI ML 3. They use multiple software development tools. However, the tools used in specific projects depend on the customer requirements. The organization has around 320 personnel, developing and providing consultancy for military systems and software projects.

The second case study (Case Study II) was executed following the completion of the first. It was designed to have three phases, corresponding to the first three stages of the PLETIN method: Context definition, process definition and process mapping. The process model for Case Study II is given in Figure 49.



**Figure 49** Process model of Case Study II

Case Study II was executed in two weeks by a single researcher. The effort spent for each phase for the case study was recorded.

The constraints for the case selection were the same as the first case study. However, there are fundamental differences between the two case studies. Table 12 summarizes these differences.

Because of the above stated differences between the two organizations, different perspectives not available in Case Study I was observed. Organization II proved to be a good match for the aim of Case Study II, and the thesis as a whole. While conforming to the constraints developed for Case Study I, Case Study II provides the following differences:

- There is no fixed tool set employed, enabling the observation of tool interactions from a wider perspective of generic process definitions.
- Although there is no fixed tool set, the tools usually employed by Organization II is almost completely different from the ones used in Organization I. This provides a different understanding from Case Study I.
- Process definitions have a less detail compared to Organization I, testing the ability of PLETIN to identify candidate integration situations from a different detail level.

**Table 12 Differences between the two target organizations**

	<b>Organization I</b>	<b>Organization II</b>
Tool set	Employs a fixed set of tools for all software development projects.	Does not have a specific tool set. Tools used can be different for each project.
Tool use	Tools are rigorously used in every software development project and constitute part of the organizational culture.	Tool use is not mandatory. The choice is on the discretion of the people responsible from the project and requests from the customers.
Process definitions	Process descriptions contain explicit description of tool interactions or include references to tool guideline documents.	Process definitions describe how the work should be done, either manually or through the use of tools. Details for tool interactions are omitted, however tool use is encouraged explicitly.
Tool guidelines	Has an extensive set of guidelines for tool use.	There are no tool guidelines (except an old guideline for configuration management tool)

To observe the PLETIN's ability of producing consistent results, the candidate integration situations identified in Case Study II was compared with the results of the Case Study I. Similarities and differences were identified.

It should be noted that a prototype implementation was not considered as a part of this case study. Implementation effort is the direct transformation of web-service and business-process definitions and could in practice be performed by expert software developers. For the analysis of the applicability of PLETIN, existence of actual web-services and business processes is not necessary. Instead, web-service and business process definitions in the form of executable process models are sufficient.

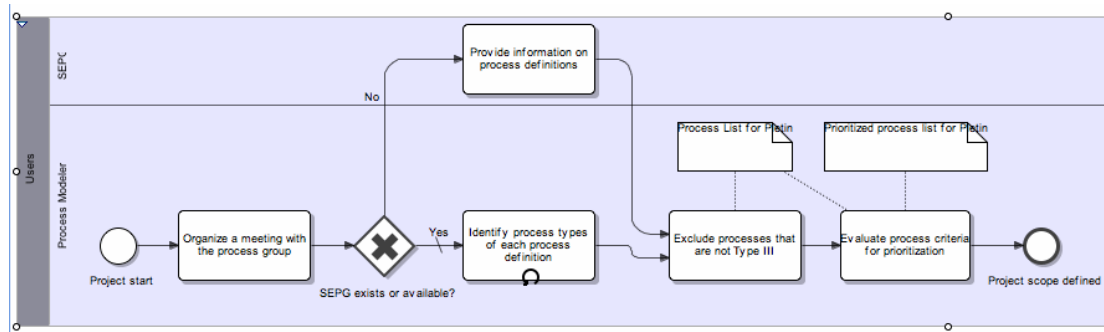
### **5.5.1. Context Definition Phase**

The aim of this phase was to define the scope of the case study. To define the scope, process definitions of Organization II were analyzed. Process definitions with no or single tool integrations were filtered out. Process model for this phase is given in Figure 50.

A meeting was held at the beginning of the case study attended by the researcher and the Head of Process Group for Organization II. The aim of the meeting was to establish mutual understanding for the conduct of the case study and acquire information on the process definitions. A secondary goal was to identify process definitions that can be excluded in bulk based on the experiences and knowledge of the process group. The following were the agenda items:

- An overview of the research performed in this thesis, along with the goals and constraints.
- An overview of the PLETIN method.
- Determination of the schedule, location and scope for the case study.
- Collection of information on the structure of organizational process definitions.
- Identification of process definitions suitable for the application of the PLETIN method.





**Figure 50 Process model of Case Study II, Phase I**

As in Organization I, process definitions were grouped into process areas similar to the CMMI process model. The following 4 process areas were selected as suitable:

- Configuration Management (CM)
- Product Development – Requirements Analysis (RA)
- Product Development – Coding, Software Unit Test and Integration (CODE)
- Product Development – Verification and Validation (VV)

Analysis of these 4 process areas defined the scope of the implementation. All process definitions in these process areas were analyzed, resulting in a list of 15 processes containing multiple tool interactions. These processes were selected for further analysis and component identification in the next phase. The list of processes is given in Table 13.

It should be noted that the structure and format of these process areas present differ slightly from the ones analyzed in Case Study I. Process definitions for Case Study II are longer, and do not present clear sub-processes that have been identified as process definitions in Case Study I. However there are sub-headings in process definitions that we have used to identify these sub-processes. Besides differences in format, significant similarities exist between the processes identified in both case studies. This was expected since software development processes interacting with tools are configuration management, requirements engineering, testing and coding. Very little interaction exists for supporting processes like project or process management.

**Table 13 List of processes selected for analysis**

<b>Process Code</b>	<b>Process Area</b>
KY-020-621	CM
UG-010-84	RA
UG-040-83	RA
UG-070-81	CODE
UG-070-82	CODE
UG-070-83	CODE
UG-070-86	CODE
UG-070-87	CODE
UG-070-89	CODE
UG-190-810	VV
UG-190-811	VV
UG-190-812	VV
UG-190-813	VV
UG-190-82	VV
UG-190-89	VV

Compared to Case Study I, the process definitions include less detail. They focus on what should be done for each process definition, contents, inputs and outputs without details. This is because of the fact that Organization II does not mandate use of tools for software development and leaves the decision to the people responsible from individual projects. Thus, actions can be executed manually in some projects, or using tools in other projects. This attribute of process definitions in Organization II is favorable being a second case study where PLETIN would be used to identify candidate tool integration situations from process definitions with less detail.

### 5.5.2. Process Definition Phase

The second phase of Case Study II focused on the development of process models based on the process definitions. 15 process definitions identified as suitable for the purposes of PLETIN in the first phase were further analyzed to identify process components. Actors, actions, tools, process flow, interactions with tools, and message contents were identified from process definitions to develop process models. The process flow for this phase is given in Figure 51.

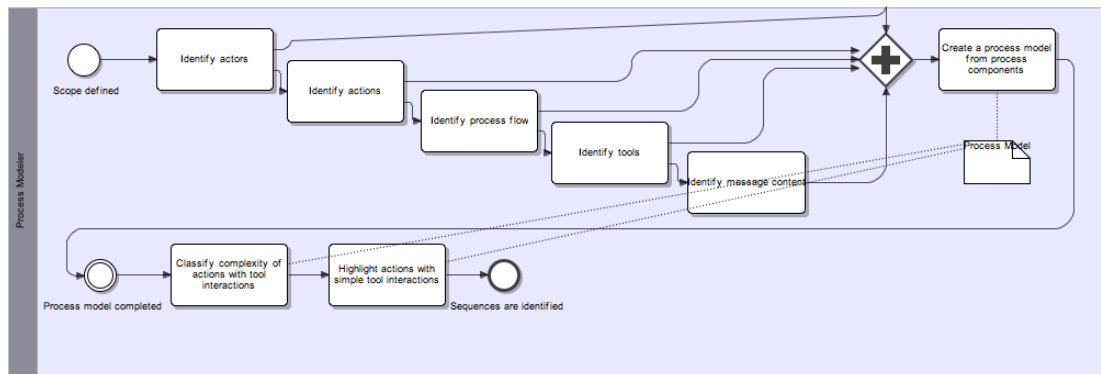


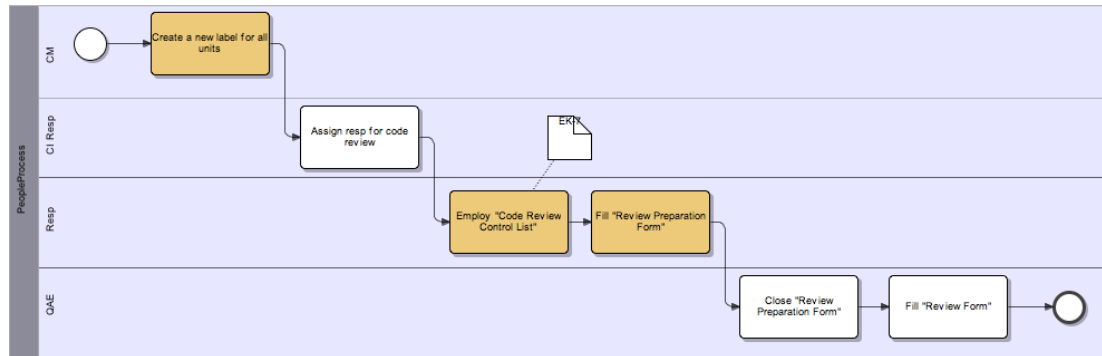
Figure 51 Process model of Case Study II, Phase II

After the process components were identified and represented as process models, tool interactions were analyzed. They were classified as simple or complex, and simple interactions were highlighted in the process model. For a detailed discussion of this classification, see Section 3.4.3.

A total of 18 process models were developed for Case Study II corresponding to the processes identified in the first phase. The process models represent the selected processes using BPMN notation. A sample process model developed in this phase is given in Figure 52. Process models developed in this phase are omitted since they were transformed to their final form in the next phase of the case study, which are given in APPENDIX F. PROCESS MODELS (CASE STUDY II).

The process models developed in this phase represent process definitions containing tool interactions and highlight the interactions that are simple, thus suitable for candidate tool integration situations. The next phase of the Case Study II, corresponding to the Process Mapping stage of PLETIN would use these process models as inputs to identify sequences of suitable tool interactions. From these

interactions web service and business process definitions would be extracted to provide the necessary infrastructure for tool integration.

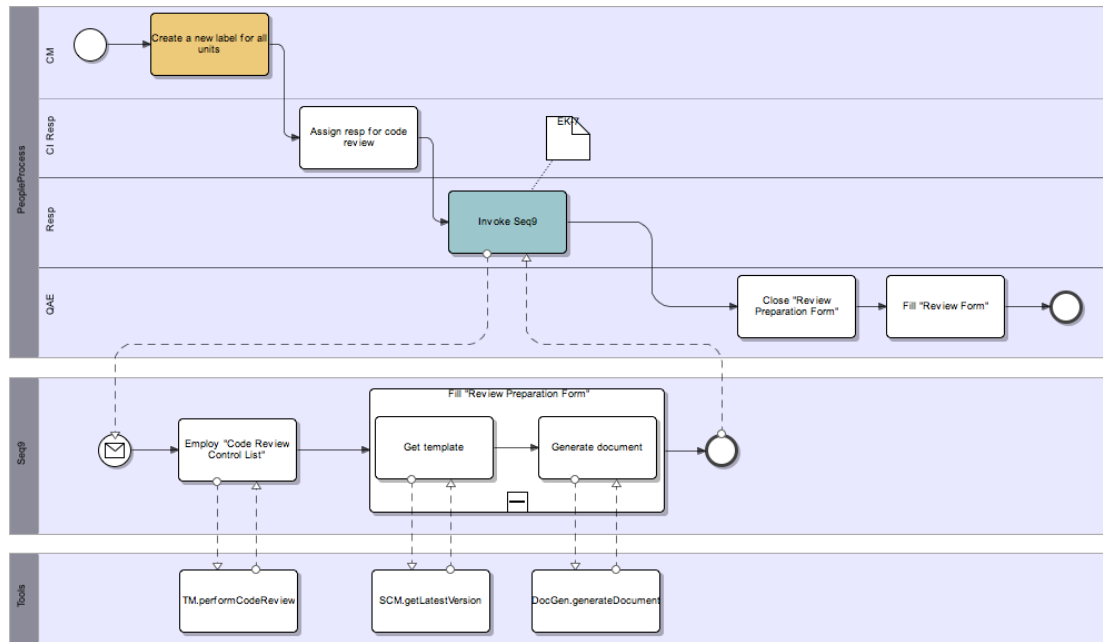


**Figure 52 Sample process model for Case Study II, Phase II (UG-070-87)**

### 5.5.3. Process Mapping Phase

Third phase of Case Study II used process models developed in the previous phase to identify candidate tool integration situations. These situations were extracted from sequences of simple tool interactions. Simple tool interactions were highlighted in the previous phase to ease their identification. In this phase of the case study, these interactions were inspected to see if they were forming up a sequence. A sequence of simple interactions are said to exist if two or more simple interactions are executed in sequence without any complex interaction or regular action in between. Interactions forming up a sequence were labeled and identified with a unique sequence number.

Actions forming up the sequences were decomposed into atomic actions. The decomposition information is given in APPENDIX D. PROCESS MAPPING (CASE STUDY II). The sequences decomposed into atomic actions were then moved onto a separate BPMN pool, representing the business process that is going to be executed by the integration infrastructure (Business Process Execution Engine). Non-atomic actions were represented as BPMN Sub-process element containing their atomic decomposition. Web-service definitions were developed based on these atomic actions. A list of web-service operations required by the user interactions are given in APPENDIX B. COMPLETE LIST OF OPERATIONS DERIVED FROM PROCESS MODELS (CASE STUDY II).



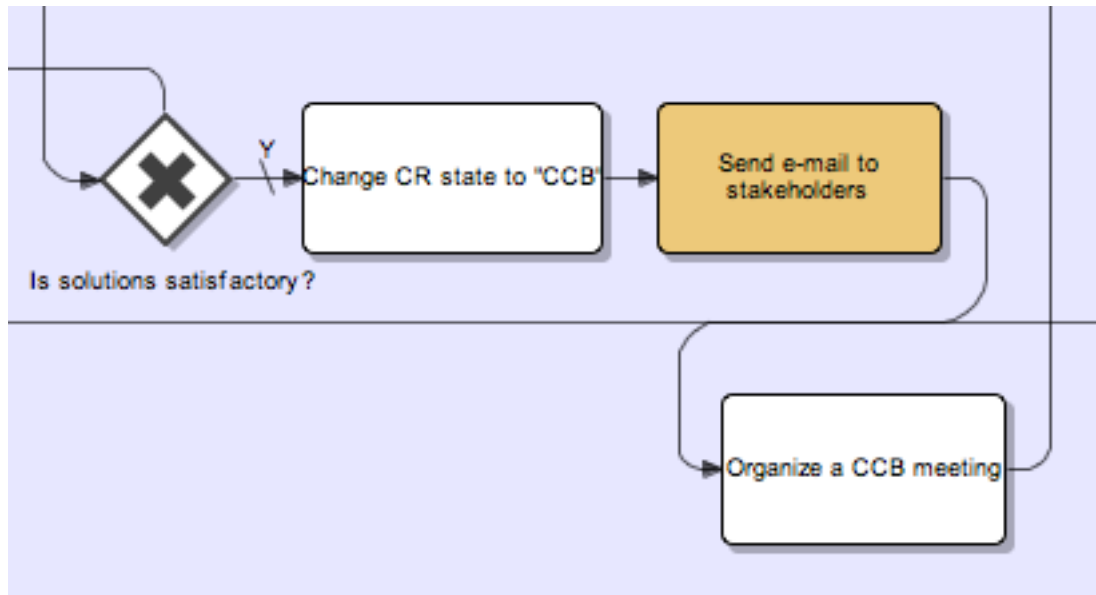
**Figure 53 Sample process model for Case Study II, Phase III**

Tasks representing web-services required from the tools were added to a separate BPMN pool in the model named “Tools”. These tasks are connected to atomic actions in the sequence using BPMN message elements to represent the information exchange between the business process executed by the integration infrastructure and services provided by the tools. A sample process model finalized in this phase is given in Figure 56.

A discussion on the comparison of the web-service definitions developed from the two case studies is given in a later section. The process model of this phase is given in Figure 53.

#### **5.5.4. Discussion on Implicit Sequences for Case Study II**

Our analysis on implicit sequences for Case Study II revealed three compound implicit sequences for Case Study II. These sequences were labeled as KY-020-62135, KY-020-62142 and KY-020-62110. KY-020-62135 and KY-020-62110 were already captured by the sequence labeled KY-020-621 because they were succeeding or preceding another simple tool interaction, thus were making up a sequence. However, KY-020-62142 was not labeled as a sequence, but identified as a standalone action. The process model for this process is given in Figure 54.



**Figure 54 Process model for KY-020-62142**

The implicit sequence consists of the software configuration management tool (SCM) sending an e-mail to relevant stakeholders. Thus the existing integration implementation between SCM and the e-mail system is employed. However, since the initial tool interaction by the user was with the SCM, and the e-mail integration is a feature of this system, it was not identified initially. However, since the process definition contains two mentions of tools a compound implicit sequence was observed (See Table 3). The sequence is easily visible when the action is decomposed (See Figure 55).

An interrupted implicit sequence was observed and labeled as UG-070-81. This sequence was discovered when the “Store code in CM” action was classified as Submit/Update/Put and the source of the information/data was found to be “Generate code” action for the UML/IDE tool.

Process definitions analyzed in Case Study II does not include any mentions other than UG-070-81 of UML modeling tools. However, analysis of the tool features reveals existing integrations with other tools like the configuration and change management tools. Unfortunately, neither the use of the UML tool, nor its integrations (in terms of storing the documents) with the configuration management tool is specified in process definitions documents. This hints several possible

unmentioned/omitted implicit sequences in process definitions. Improvement of these process definitions through observation and discovery is left as future work.

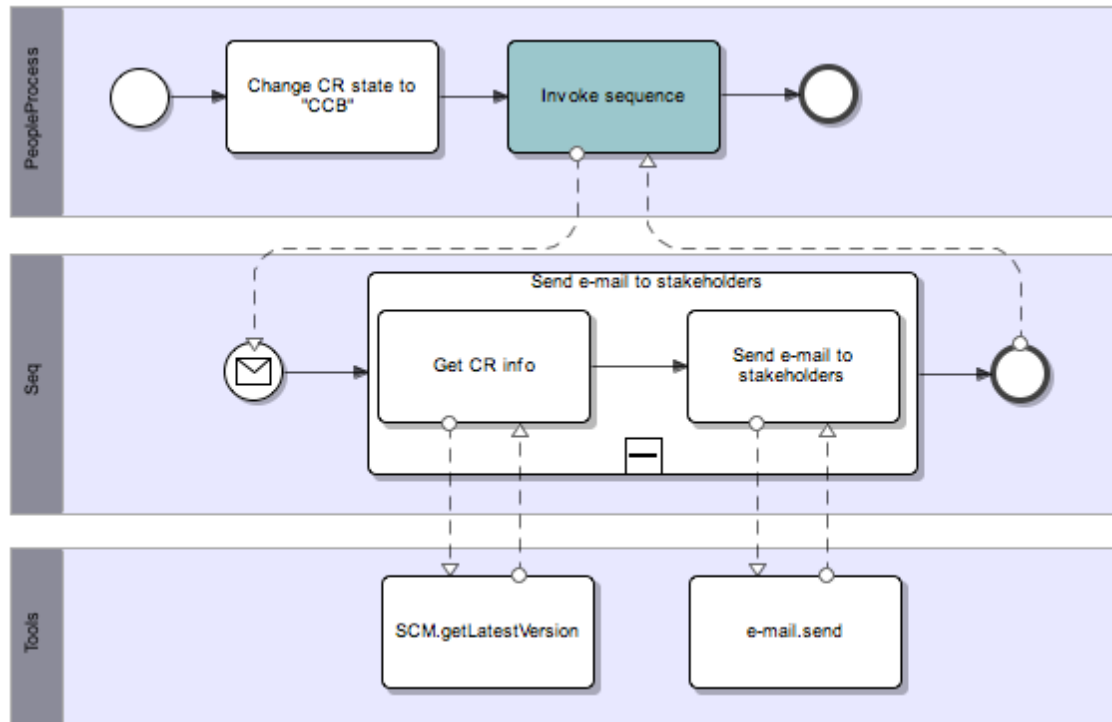


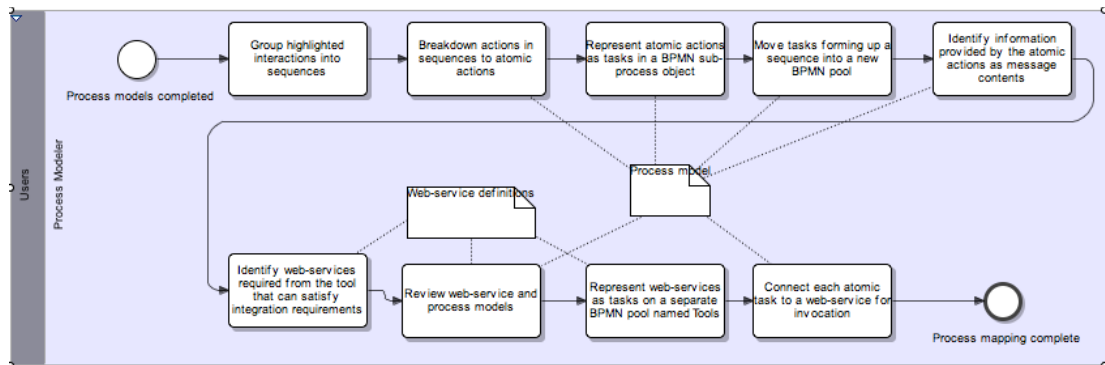
Figure 55 Sequence KY-020-62142 decomposed

## 5.6. Results for Case Study II

Case Study II was conducted in two weeks following the completion of Case Study I. Following the initial meeting with the process group, process definitions from 4 process areas were analyzed. A total of 15 process definitions were classified as suitable for the purposes of the PLETIN method. These 15 process definitions were represented as 18 separate process models.

Analysis of tool interactions in these process models revealed 25 sequences, consisting of 58 invocations of 14 different operations provided by 7 different systems. These systems are: Change Management Tool (ChM) IBM Rational ClearQuest, Software Configuration Management Tool (SCM) IBM Rational ClearCase, Test Management Tool (TM) HP Mercury Quality Center, UML Modeling Tool (UML) Rational Rose, File System, E-mail System, and Document Generator (DocGen). Requirements Management Tool (RM) IBM Doors is also used

in the organization, however sequences derived from process definitions did not contain any interactions with it. The list of all operations is available in APPENDIX B. COMPLETE LIST OF OPERATIONS DERIVED FROM PROCESS MODELS (CASE STUDY II).



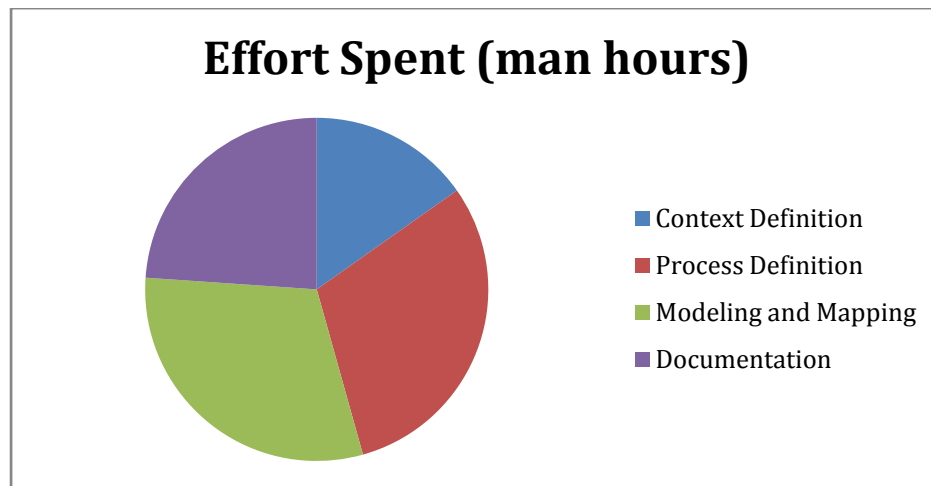
**Figure 56 Process model of Case Study II, Phase III**

We should note that:

1. In the scope of Case Study II, DocGen represents an abstract set of functionality containing all document generation capabilities of software development tools.
2. Unlike Case Study I, system classification in Case Study II does not contain a Project Repository (PR) where information regarding the projects is stored. During Case Study I, a need for a system like PR was revealed in the second iteration of web-service definition when existing functionality of the tool set was compared to initial definitions. In Case Study II, details regarding the tool set was omitted due to several constraints, thus detailed information for the existing tool set was not available. This prevented the detailed understanding of the PR system, or whether it was necessary at all.

Case Study II was also performed by a single researcher. During the two weeks of case study conduct, a total of 23 man hours of effort was spent. The effort distribution for case study activities is given in Figure 57:





**Figure 57 Effort distribution for Case Study II**

### **5.7. Validity Threats**

The multiple case study design requires a software development organization with defined and mature processes employing multiple supporting tools. Such organizations are not plenty in existence. However, we have designed the case studies and the resulting method so that, they can be applied to organizations having an intention to understand their processes and develop process definitions. It would be greatly beneficial to perform this case study, or apply the PLETIN method concurrently with process development, process improvement or process modeling efforts. This way the outputs of the main effort can be consumed for the case study and/or the PLETIN method.

A single researcher who had professional experience with the tools and processes employed in the target organization performed the case studies. This would mean more effort for a researcher with no existing background on tools, processes and modeling to conduct a similar case study or apply the PLETIN method. Since the number of processes was small and the method was still under development, a collaborative work with the employees of the target organizations was not considered. However in a larger setting with multiple divisions, more complex process library and many tools, extensive help from the organization may be sought in process analysis and implementation. PLETIN method does not have inherent complexity in its execution. It relies on the understanding of the processes and user

interactions. Thus an initial briefing to fellow modelers would be recommended and satisfactory. Also, after the modeling effort is completed, consistency of the processes must be checked before deployment.

The application of the PLETIN method in two different settings proved its flexibility. Although the constraints for case selection in both cases were the same, two organizations proved to present significant differences, negating the possible bias resulting from case selection.

## **5.8. Discussion**

The most significant output of this multiple case study effort is the PLETIN method which is developed during the execution of Case Study I, and implemented in both cases.

The PLETIN method has been developed parallel to the execution of Case Study I and contains activities from all its phases. The first phase of Case Study I has evolved into the “context definition” stage of the PLETIN method. The second phase of Case Study I has evolved into the “process definition” stage while the last phase has evolved into the “process mapping” and “process execution” stages.

Besides the development of the method, the following observations were done during Case Study I:

1. Tools are used extensively in software development.
2. Tool integration is insufficient and should be improved.
3. A method is required, to identify and implement the missing integration-tuples directly from process definitions.

At the beginning of Case Study I we have analyzed the interactions of users from tools, taking a tool integration perspective. We have found out that tools are used extensively and frequently to support tool interaction. This proved a strong dependency of users and processes for the tools.

Assured with the extent of tool use and the existence of a strong dependency on the tools, we focused on the existence and significance of issues with tool interaction.

Observation of these issues would indicate increased benefits from tool integration. We have devised a method to identify candidate integration situations, what we call integration-tuples. We have created integration maps of the existing situation and what the processes require. This gap analysis provided us information on how the requirements of process definitions and users were not satisfied in terms of tool integration.

Besides the obvious lack of integration facilities, our correspondence with the users and the process owners suggested the following problems which are in accordance with [1], [14], [38], and [54]:

1. Lack of a standard framework for tool integration or an integration infrastructure forces organization to choose tools based on integration facilities provided rather than overall features. This results in vendor-dependency (or vendor lock-in) through tool suites. Organizations are dictated to use inferior tools to satisfy integration requirements.
2. Implementations of integration-tuples from vendors or 3<sup>rd</sup>-parties are fragile, version-dependent, and volatile. These implementations are point-to-point and do not provide an all-encompassing solution, becoming unmanageable in time.
3. Changes in tools (for example upgrades) result in these integration-tuples to become obsolete, which in effect reduces functionality and frustrates users. Discovery of recently added functionalities requires extra effort from the organization and they largely remain unexploited.

The next step taken to provide a solution to these issues was to devise a method to close the gap between the integration requirements and tool facilities. The method aimed to convert integration-tuples presented as process models into implementations. This corresponds to the last two stages of the PLETIN method we have developed.

The PLETIN method is easy to implement because it requires only the process definitions and information on the tools used from the organization. A single

individual, proficient in the domain was able to undertake the modeling and mapping effort in both cases. The implementation effort for Case Study I provided a prototype application. In practice, this effort can be distributed to developers skilled in related technologies and executed in parallel in a much shorter time.

The results of Case Study I showed that PLETIN was easy to implement and it provided practical results that can be realized without much effort. However, since the method itself was developed based on this case, the results of the implementation experience may be biased. Case Study II was designed and conducted to provide independent observations for the implementation of PLETIN in a separate environment. Case Study II verified the applicability of PLETIN in a different organization with a different tool set, and a different approach to tool use.

The integration-tuples identified by PLETIN are independent of the existing toolset. However, in the later stages of the method, tuples are mapped to interfaces provided by the tools to develop integration implementations. These mappings can be performed to any tool providing interfaces or services for customizations. This way, organizations are not enforced to use any tool because of the integration features it provides. Rather, organizations can have tools that are suitable for their processes and implement integration-tuples between them using the PLETIN method as business processes and related web-services. Any tool can be plugged into the system anytime. The only requirement is the mapping of the tool services and organizational process requirements. This provides flexibility in tool selection and tool interchangeability for the organization.

Although the structure of the process definitions, choice of tool sets and their use were different, Case Study II provided similar results to Case Study I. Tool interactions were identified from process definitions providing integration opportunities. These candidate integration situations were represented in process models, based on which web-service and business process definitions required for the development of a tool integration framework were derived.

Case Study II provided a less diversified set of integration situations. This was expected since the process definitions encouraged the use of tools rather than

enforcing. Also, process definitions did not include any guidelines on tool use which made up the largest portion of sequences identified in Case Study I.

Of the 14 operations identified in Case Study II, 11 operations (78%) were already identified in Case Study I, or defined in a very similar fashion. This overlap is quite significant and confirms our expectations contrary to differences between two cases. Such an observed overlap between two organizations with different tool sets and different approaches to software development tool use encourages the possibility of the development of standard tool interfaces for tool integration based on organizational process definitions. PLETIN can be used by organizations with different attributes to identify tool integration requirements imposed by organizational processes.

The conduct of the Case Study II took considerably less time (20%) compared to Case Study I. The difference results from the experience gained in the first implementation, coupled with less number of interactions, less process detail in terms of tool interaction and the lack of the need for a prototype implementation. This would be beneficial for organizations with tight budget and personnel constraints, or external process consultants with limited schedules. PLETIN provides a direct guideline for analyzing the tool requirements of an organization.

The implementation of integration-tuples as automatically executed processes that are normally performed manually by users brings all the benefits of process automation including: faster execution, less manual effort, less errors, visibility, better measurements, easier to change.

In our case studies, we have observed that the execution frequencies for the sequences identified range from several times per day to once per year. For sequences that are executed several times every day, the effort saved by the automation of actions normally performed manually more than compensates the effort spent for PLETIN. Besides the effort saved, automatic execution of processes prevents operator errors, or steps missed for menial activities. This benefit of automated processes is observed when tools in an organization is integrated using PLETIN, and the tool cooperation is no longer maintained manually.

The components used for the PLETIN method are freely available. The modeling notation used is BPMN, which is an open specification and many tools support it. During the case studies we have used Intalio products [37] for both BPMN modeling and process execution. Intalio BPMN Suite Community Edition is a free tool that satisfied all our needs. Other process execution engines and modeling tools can be used with almost no modifications to the PLETIN method and case study conduct.

This effort can be or even recommended to be undertaken as part of, or parallel with an existing process definition, improvement or modeling project. Both projects can benefit the other. For example, process models developed during the case studies can be used as a basis for process communication and improvement efforts while providing input for the execution of integration-tuples.

Since software processes are software too as stated by Osterweil [52], they are subject to change [9]. To manage this change in software processes, PLETIN provides easy deployment of process models to execution engines. Thus any change in process models is quickly reflected on the execution of the processes, i.e. implementation of integration-tuples.

The outputs of the case studies including the integration-tuple definitions and web-service specifications developed from the organizational processes can be combined with the results of other similar case studies to digest a knowledge base of integration requirements across the industry. This information can guide, or even force vendors to develop tools complying with these requirements. Such tools would prove to be interoperable and interchangeable because of standard interfaces they support. This information on the requirements, the operations required and messages interchanged can be used to develop an understanding of the software engineering domain in the form of a domain ontology.

## **CHAPTER 6**

### **CONCLUSIONS**

Tool integration is a high priority topic during tool selection for software development organizations. An integrated tool set is sought to produce better products, easier and cheaper through better execution of processes. Organizations should be able to develop a competent tool set that is economically feasible while satisfying all the requirements of the organizational processes. They should be able to choose either best-in-class tools or tools that provide adequate functionality at an acceptable cost.

Unfortunately, the integration functionality offered by state-of-the-art software development tools are either biased towards tools of the same vendor to establish a tool suite, or are bilateral, version-dependant, hard to maintain and fragile.

None of the many efforts available in the literature has been widely accepted in practice. We believe the problem is based on the fact that these efforts only provide guidelines, architectural models and constraints for tools to be developed. However, there is already a market for software development tools and organizations already own some tools. An approach that can provide integration facilities for the existing tools is necessary.

Another facet of the problem is the approach taken by previous efforts, focusing on the technicalities of integration like which data to share, how to store, translate and

manipulate data, how to notify other tools or how to publish services. We believe that rather than asking “how to integrate?”, we should ask for “what to integrate?” and “when to integrate?”.

To answer these questions, a method named PLETIN has been developed to identify which tool integration facilities are required by the organizational processes. In the proposed method, process models are developed to visualize process definitions. The integration requirements extracted from the process models are used to define and build custom interfaces for the tool set employed by the organization. Business processes are developed from process models, which mimic the manual actions performed by users. These business processes consume the interfaces developed for the tools when executed automatically. User actions are performed by the integration infrastructure on behalf of them and tools are integrated based on the requirements derived from organizational processes.

The PLETIN method is suitable for organizations that employ multiple tools for software development and have problems with their existing tool set in terms of integration. PLETIN relies on process definitions of the organization so existence of mature process definitions is a must.

The method has been based on the knowledge gained from a case study designed to observe the state of tool use and issues of tool integration. The case study was performed in a software development branch of a research organization. This branch develops software for military and civilian systems. To support their development efforts, they utilize multiple tools and have process definitions in place, evaluated to be CMMI ML3.

The case study proved us that software development is highly dependent on tool use, and several issues stemming from tool integration have been observed. Case study results prove that software development processes in our target organization require a more integrated tool set and can exploit the integration functionalities if they exist.

The PLETIN method developed in conjunction with the first two phases of the case study was applied on the same case to develop custom implementation integrations. We have found out that the PLETIN method was easy to implement, and helped us



rapidly identify candidate tool integration situations. Based on the outputs of the method, an actual prototype implementation was completed in short notice with relatively low resources.

The prototype implementation enabled automated execution of action sequences, normally performed manually by the users. The actions were delegated to a business execution engine. As a result, several actions similar to a documentation sequence consisting of two baseline operations on two tools, obtaining a file and generating a document was completely defined as an automatically executable process. This lets users perceive the sequence to be cooperatively operated by the tools, as if they are tightly integrated over a process definition. Normally, such sequences are menial, time-consuming and error prone. Critical steps like putting the document under configuration management are easily forgotten. With the use of PLETIN, these sequences are defined as automated processes. They are performed reliably, quickly, and without errors by the integration framework based on a business process engine every time they are initiated.

PLETIN was used in a second case study to validate its applicability in a different setting. With much less effort, similar candidate integration situations were identified from process definitions of an organization with a different tool set, and different policies for tool use.

The PLETIN method has significant practical value to organizations since it is directly applicable to existing tools and processes. Organizations can develop custom integration solutions satisfying the requirements of their software development processes.

Since organizational requirements from the tools are identified, different tools fulfilling these requirements can be identified and employed. Custom wrapper code can be developed based on the organizational requirements and service definitions to incorporate tools into the environment. Tools can be interchanged or new tools can be incorporated to the tool set. Existing functionality can be modified to support the new tools.

PLETIN is developed using open standards and technologies like BPMN, BPEL4WS, and web-services. These technologies are widely used in practice and available from different sources. Being ubiquitous, they are widely supported by the industry. There are many process execution solutions supporting BPMN and BPEL4WS. Tool vendors provide web-services based interfaces for customization of their tools. PLETIN, based on these standards and technologies enables the integration of a wide variety of tools.

PLETIN extracts service and process definitions from organizational processes. This information is used to define the interfaces between the organizational processes and the tools used during software development. These interfaces can be implemented by any tool providing interfaces for customizations. This enables organizations to choose tools that best suit their requirements and incorporate them into their environments. Tools can be interchanged with other tools providing (or customized to provide) services required by the organizational processes.

After the identification of services organizations require from tools, PLETIN provides a method for the mapping of these services to the interfaces tools provide. Using PLETIN, custom interfaces supporting the requirements of organizations can be built. Organizations can build custom interfaces for their existing tools. This way existing tools can be integrated and support the requirements of the organizational tools.

Information extracted from process definitions of different organizations can be used to digest industry-wide process requirements from tools. This information is useful for understanding user interactions with tools and can be used to provide integration points for future releases of tools.

PLETIN in its current state has limitations such as: requires the existence of process definitions, uses a subjective classification scheme for the complexity of tool interaction, omits “Black Box” tools, not generalized to other tools and notations, does not incorporate the whole domain but can be generalized with further case studies, does not take other events generated in the environment into account, omits complex data mappings, does not consider already existing (legacy) or new

integration features (for a detailed description, see Section 3.10). These limitations provide future research directions in the tool integration domain to develop a full-featured tool integration solution with practical importance.

There is also much information that can be obtained from the analysis of other organizations to identify common requirements for integration. This knowledge can be re-used in process definitions across the industry. Even future tool designs can benefit from these integration requirements. This knowledge can also be used to develop an ontology for the software domain that includes messages, objects and actions used during software development.

The method developed for this thesis enables parts of software development process to be delegated to a business process execution engine and mapped to services provided by the tools. This way, they can be executed on a business process execution engine without manual intervention [48], [53], [61]. The execution engine performs the actions on behalf of the users with respect to the process model while providing an integration infrastructure for the tools. Actions normally performed by users are executed automatically by this infrastructure, thus automated. Process automation efforts are undertaken to increase quality, efficiency, reliability of the processes while decreasing costs and errors. By partial automation through tool integration, processes are executed faster and with fewer errors due to elimination of human-errors and intervention.

## REFERENCES

- [1] Altheide, F. & Dörfel, S. (2003). An architecture for a sustainable tool integration. In Dörr H, Kanzleiter J. (Ed.), ESEC/FSE workshop on tool integration in system development (pp. 29-32).
- [2] D'Ambrogio, A. and Iazeolla, G. (2005). Metadata-driven design of integrated environments for software performance validation. *Journal of Systems and Software*, 76( 2 ), 127-146
- [3] Anderson, M. J. & Bird, B. D. (1993). An evaluation of PCTE as a portable tool platform. In *Proceedings of the Software Engineering Environments Conference* (pp. 96-100).
- [4] Apache ODE BPEL engine. (n.d.). Retrieved January 23, 2010 from: <http://ode.apache.org/>
- [5] Apache Tomcat. (n.d.). Retrieved January 23, 2010 from: <http://tomcat.apache.org/>
- [6] Application Lifecycle Framework (ALF). (n.d.). Retrieved January 2009 from Eclipse Web site: <http://www.eclipse.org/alf>
- [7] Arsanjani A. (2004). Service-oriented modeling and architecture. Retrieved January 23, 2010 from <http://www.ibm.com/developerworks/library/ws-soa-design1/>
- [8] Bandinelli, S., Fuggetta, A., Lavazza, L., Pietro Picco, G. (1994). Combining control and data integration in the SPADE-1 process- centered software engineering environment. In *9th International Software Process Workshop* (pp. 96–99).

- [9] Barthelmeß, P. (2003). Collaboration and coordination in process-centered software development environments: a review of the literature. *Information and Software Technology*, 45(13), 911-928.
- [10] Belkhatir, N. & Estublier, J. (1986). Protection and cooperation in a software engineering environment. In Conradi et al. (Ed.). *Proceedings of an International Workshop on Advanced Programming Environments* (pp. 221–229). Springer-Verlag London, UK.
- [11] Bergstra, J. A. & Klint, P. (1998). The Discrete Time ToolBus – a software coordination architecture. *Science of Computer Programming*, 31(2-3), 205-229.
- [12] Borland StarTeam (n.d.). Retrieved January 23, 2010 from:  
<http://www.borland.com/us/products/starteam/index.html>
- [13] Brown, A.W. (1993). Control Integration Through Message Passing in a Software Development Environment. *Software Engineering Journal*, 8(3), 121-131.
- [14] Brown, A. W., Feiler, P. H., Wallnau, K. C. (1991). Understanding Integration in a Software Development Environment. (Tech. Rep. No: CMU/SEI-91-TR-31, ADA248119). Software Engineering Institute, Carnegie Mellon University.
- [15] Brown, A. W., Feiler, P. H., Wallnau, K. C. (1993). Past and future models of CASE integration. In *Proceedings of Fifth International Workshop on Computer-Aided Software Engineering, 1992* ( pp.36-45).
- [16] Business Process Execution Language for Web Services (BPEL4WS) version 1.1. (2002). Retrieved January 23, 2010 from:  
<http://www.ibm.com/developerworks/library/ws-bpel/>
- [17] Business Process Modeling Notation, V1.1 (BPMN), OMG Available Specification. (2008). Retrieved January 23, 2010 from  
<http://www.omg.org/spec/BPMN/1.1/PDF>

- [18] Capability Maturity Model Integration (CMMI). (n.d.). Retrieved January 23, 2010 from <http://www.sei.cmu.edu/cmmi/>
- [19] Chen M. & Norman, R.J. (1992). A framework for integrated CASE. *IEEE Software*, 9(2):18-22.
- [20] Christie, A. (1994). A Practical Guide to the Technology and Adoption of Software Process Automation. (Tech. Rep. No: CMU/SEI-94-TR-007). Software Engineering Institute, Carnegie Mellon University.
- [21] CMMI for Development, Version 1.2. (2006). Retrieved January 23, 2010 from:  
<http://www.sei.cmu.edu/publications/documents/06.reports/06tr008.html>.
- [22] Corradini, F., Mariani, L., Merelli, E. (2003). An agent-based layered middleware as tool integration. In the Workshop on Tool Integration in System Development (TIS 2003) in the 9th European Software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003).
- [23] Earl, A. (1990). Principles of a Reference Model for Computer Aided Software Engineering Environments. In Proceedings of the international workshop on environments on Software engineering environments (pp. 115-129). Springer-Verlag, New York, USA.
- [24] Eclipse Foundation. (n.d.). Retrieved January 23, 2010 from <http://www.eclipse.org/>
- [25] Eclipse Application Lifecycle Framework (ALF) project- Project Termination Review (2008) Retrieved January 23, 2010 from <http://www.eclipse.org/project-slides/Eclipse%20ALF%20Termination%20Review%20Nov%202008%20v02.pdf>
- [26] Erturkmen, K.A., Demirors, O. (2009). Integration of CASE Tools to Software Processes: A Case Study. In Industrial Proceedings of 16th

European Systems and Software Process Improvement and Innovation Conference (EuroSPI'2009) (pp:11.1-11.2).

- [27] Erturkmen, K.A., Demirors, O. (2010). Software Development Processes-Based Tool Integration Using the PLETIN Method: A Case Study. Submitted for Publication.
- [28] European Computer Manufacturers Association. (1993). Reference Model for Frameworks of Software Engineering Environments. Retrieved January 23, 2010 from: <http://www.ecma-international.org/publications/techreports/E-TR-055.htm>
- [29] Flatscher, R.G. (2002).Metamodeling in EIA/CDIF—Meta-Metamodel and Metamodels. *ACM Transactions on Modeling and Computer Simulation*, 12(4), 322–342.
- [30] Freude, R. & Königs, A. (2003). Tool integration with consistency relations and their visualization. In the Workshop on Tool Integration in System Development (TIS 2003), in the 9th European Software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003)
- [31] Gautier, B., Loftus, C., Sherratt, E., and Thomas, L. (1995). Tool integration: experiences and directions. In Proceedings of the 17th international Conference on Software Engineering (Seattle, Washington, United States, April 24 - 28, 1995). ICSE '95 (pp. , 315-324). ACM, New York, NY.
- [32] Gisi, M.A. Kaiser, G.E. (1991). Extending a Tool Integration Language. In Proceedings. First International Conference on the Software Process( pp.218-227)
- [33] Grundy, J., Apperley, M., Mugridge, R., Hosking, J. (1998). Tool Integration, Collaboration and User Interaction Issues in Component-Based Software Architectures. In Proceedings of the Technology of Object-Oriented

Languages and Systems, November 23 - 26, 1998 (pp: 299). IEEE Computer Society, Washington, DC.

- [34] Guo, B., Shen, Y., Xie, J., Wang, Y., Xiong, G.Z. (2004). A kind of new ToolBus model research and implementation. ACM SIGSOFT Software Engineering Notes, 29(2), 5
- [35] Hansen, K.M. (2003). Activity-centred tool integration. In the Workshop on Tool Integration in System Development (TIS 2003), in the 9th European Software Engineering Conference and 11th ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003).
- [36] Harrison, W., Oshser, H., and Tarr, P. (2000). Software engineering tools and environments: A roadmap. In 22nd International Conference on Software Engineering, Future of Software Engineering Track (pp. 261-277). Limerick Ireland. ACM Press.
- [37] Intalio BPM Suite. (n.d.). Retrieved January 23, 2010 from:  
<http://www.intalioworks.com/products/bpm/opensource-edition/>
- [38] Integration and Interoperability of Application Lifecycle Management tools. (n.d.). Retrieved January 23, 2010, from [http://www.docstoc.com/docs/-976078/Tech\\_Application\\_Lifecycle\\_Management\\_tools](http://www.docstoc.com/docs/-976078/Tech_Application_Lifecycle_Management_tools)
- [39] ISO/IEC 15504 Standard. (2007). Retrieved January 23, 2010 from:  
<http://www.isospice.com/categories/ISO{47}IEC-15504-Standard/>
- [40] Java Enterprise Edition (J2EE) (n.d.). Retrieved January 23, 2010 from  
<http://java.sun.com/javaee/>
- [41] Java Servlet Technology (n.d.). Retrieved January 23, 2010 from  
<http://java.sun.com/products/servlet/>
- [42] JavaServer Pages Technology (n.d.). Retrieved January 23, 2010 from  
<http://java.sun.com/products/jsp/>
- [43] Kaiser, G.E., Barghouti, N.S., Sokolsky, M.H. (1990) Preliminary experience with process modeling in the MARVEL software development



- environment kernel. In Proceedings of the Twenty-Third Annual Hawaii International Conference on System Sciences: Vol: 2 (pp. 131-140).
- [44] Kapsammer, E., Reiter, T., Schwinger, W. (2006). Model-Based Tool Integration - State of the Art and Future Perspectives. In Proceedings of the 3rd International Conference on Cybernetics and Information Technologies, Systems and Applications (CITSA 2006), Orlando, USA. 2006. (pp: 20-23)
- [45] Lundell, B. & Lings, B. (2004), Changing perceptions of CASE technology. *Journal of Systems and Software* 72 (2), 271-280.
- [46] Mi, P. and Scacchi, W. (1992). Process Integration in CASE Environments. *IEEE Software*, 9( 2), 45-53
- [47] OMG MetaObject Facility (MOF). (n.d.). Retrieved January 23, 2010 from: <http://www.omg.org/mof/>
- [48] Papazoglou, M. P. & Heuvel, W. V. (2006). Service-oriented design and development methodology. *International Journal of Web Engineering and Technology*, 2(4), 412-442.
- [49] Oberndorf, P. A. (1998). The Common Ada Programming Support Environment (APSE) Interface Set (CAIS). *IEEE Transactions on Software Engineering*, 14(6), 742-748.
- [50] Object Management Group, Common Object Request Broker Architecture (CORBA). (n.d.). Retrieved January 23, 2010 from: <http://www.omg.org/corba>
- [51] Object Management Group, Open Tool Integration Framework Request for Proposal (OTIF). (n.d.). Retrieved March 21, 2009 from: <http://www.omg.org/docs/mic/04-08-01.pdf>
- [52] Osterweil, L. (1987). Software processes are software too. In Proceedings of the 9th international Conference on Software Engineering (pp. 2-13). IEEE Computer Society Press, Los Alamitos, CA.

- [53] Owen, M., Raj, J. (2004). BPMN and Business Process Management. Retrieved January 23, 2010 from [http://www.bpmn.org/Documents/6AD5D16960.BPMN\\_and\\_BPM.pdf](http://www.bpmn.org/Documents/6AD5D16960.BPMN_and_BPM.pdf)
- [54] Rader, J., Morris, E.J., Brown, A.W. (1993) An investigation into the state-of-the practice of CASE tool integration. In the Proceedings of Software Engineering Environments Conference (pp. 209-221).
- [55] Rony G. F. (2002). Metamodeling in EIA/CDIF---meta-metamodel and metamodels. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 12(4), 322-342.
- [56] Schürr, A., Dörr, H. (2005). Introduction to the special SoSym section on model-based tool integration. *Journal on Software and Systems Modeling (SoSym)*, Springer-Verlag, 4(2).
- [57] Thomas, I. & Nejme, B.A. (1992). Definitions of Tool Integration for Environments. *IEEE Software*, 9 ( 2), 29-35.
- [58] Unified Modeling Language (UML). (n.d.). Retrieved January 23, 2010 from: <http://www.uml.org/>
- [59] Valetto, G. and Kaiser, G. E. (1995). Enveloping Sophisticated Tools into Computer-Aided Software Engineering Environments. In Proceedings of the Seventh international Workshop on Computer-Aided Software Engineering (July 10 - 14, 1995) (pp. 40). IEEE Computer Society, Washington, DC.
- [60] Wasserman, A.I. (1989). Tool integration in software engineering environments. In Long, F. (Ed.), *The International Workshop on Environments (Software Engineering Environments)*. In *Lecture Notes in Computer Science*, vol. 647. (pp. 137–149). Springer-Verlag, Berlin, Chinon, France.
- [61] White, S. (2004). Introduction to BPMN. Retrieved January 23, 2010 from [http://www.bpmn.org/Documents/Introduction\\_to\\_BPMN.pdf](http://www.bpmn.org/Documents/Introduction_to_BPMN.pdf)

- [62] Wicks, M.N. (2006). Tool integration in software engineering: an annotated bibliography. (Tech. Rep. No: HW-MACS-TR-0041). HeriotWatt University.
- [63] Wicks, M. and Dewar, R. (2007). A new research agenda for tool integration. *Journal of Systems and Software*, 80(9), 1569-1585.
- [64] Web Services Business Process Execution Language Version 2.0, OASIS Standard. (2007). Retrieved January 23, 2010 from: <http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html>
- [65] Web Services Description Language (WSDL) 1.1. (2001). Retrieved January 23, 2010 from: <http://www.w3.org/TR/wsdl>
- [66] XML Metadata Interchange (XMI) 2.1.1 (2007). Retrieved January 23, 2010 from: <http://www.omg.org/technology/documents/formal/xmi.htm>
- [67] Extensible Markup Language (XML). (n.d.). Retrieved January 23, 2010 from: <http://www.w3.org/XML/>
- [68] Zhao, X., Chan, K., Li, M. (2005). Applying agent technology to software process modeling and process-centered software engineering environment. In Haddad, Hisham, Liebrock, Lorie M., Omicini, Andrea, Wainwright, Roger L. (Eds.), SAC. ACM (pp. 1529–1533).

## APPENDICES

### APPENDIX A: COMPLETE LIST OF OPERATIONS DERIVED FROM PROCESS MODELS (CASE STUDY I)

SCM	RM	PR
activateWorkflow	assignReqTypeToProject	lookup
createCRFilter	assignSecurityProfileToProject	
createFileFilter	assignUserGroupToProject	<b>E-mail</b>
createFolderAccessRights	assignUserToProject	send

createLabel	createBaseline	
createLink	createGlossary	<b>File</b>
createProject	createProject	appendFile
createProjectAccessRights	createTrace	createFolder
createReports	generateDocument	Execute
createServerAccessRights	importProject	extractPackage
createStatusFilters	publishRequirements	renameFolder
createTask		
createUser	<b>UML</b>	
createUserGroup	generateDocument	
freezeLabel	generateCode	
getItem		
getLatestVersion		
getLinkedItems		
getLinkedItemStatus		
setProjectAccessRight		
setStatus		
updateFile		

**APPENDIX B: COMPLETE LIST OF OPERATIONS  
DERIVED FROM PROCESS MODELS (CASE STUDY II)**

<b>SCM</b>	<b>ChM</b>	<b>TM</b>
createFileAccessRights	createTask	executeTest
createLabel	getLinkedItems	checkUnitTest
getLatestVersion	createLink	performCoderReview
updateFile		<b>UML/IDE</b>
		generateCode
<b>DocGen</b>	<b>E-mail</b>	<b>File</b>
generateDocument	send	appendFile

## **APPENDIX C: PROCESS LIST (CASE STUDY I)**

<b>PP</b>	<b>Project Planning</b>		
Project planning	Assign key project leadership positions		Type 0
	Designate project planning group		Type 0
	Analyze project requirements		Type 0
	Develop SPMP	Establish the project's defined process	Type IV
		Develop WBS	Type IV
		Develop estimates	Type IV
		Plan for needed knowledge and skills	Type 0
		Plan for project resources	Type 0
		Establish budget and schedule	Type 0
		Plan for configuration management	Type 0
		Plan for quality assurance	Type 0
		Plan for risk management	Type 0
		Plan for V&V	Type 0
		Plan for data management	Type 0
		Plan stakeholder involvement	Type 0
		Integrate plans	Type 0
	Plan for measurement activities		Type IV
	Obtain commitment to project management plan		Type IV
	Conduct project progress reviews		Type IV
	Conduct milestone progress reviews		Type IV
	Initiate and manage corrective action		Type IV
	Revise SPMP		Type IV
Close out project			Type IV
<b>CM</b>	<b>Configuration Management</b>		
Develop and maintain configuration management plan			Type IV
Manage the software configuration management activities			Type 0
Establish configuration management system	Establish CM mechanisms		Type IV
	Establish hardware environment		Type 0
	Establish software environment		Type IV
	Establish initial product directory structure		Type III
Provide CM training			Type 0
Perform configuration identification	Select configuration items		Type 0
	Identify baselines		Type 0
Perform configuration control			Type II
Perform configuration status accounting			Type IV
Perform configuration audit			Type IV
Perform data management			Type IV
Perform build management			Type III
Perform release management			Type IV
Package and deliver the product			Type 0
Project closure			Type III
<b>RE</b>	<b>Requirements Engineering</b>		
Preparation			Type III
Requirements development	Develop customer requirements	Elicit needs	Type IV
		Establish customer requirements	Type IV
		Review customer requirements	Type III
		Validate customer requirements	Type III
	Develop software requirements	Establish software requirements	Type IV
		- Define product components and interface requirements	Type IV
		- Establish software requirements specification document	Type II
		- Review software requirements	Type IV
		Validate software requirements	Type III
	Manage changes to requirements and inconsistencies between requirements and work products		Type IV
Requirements management			Type IV
<b>TS</b>	<b>Technical Solution</b>		
Develop the design	Develop screening and selection criteria for design alternatives		Type IV
	Develop high-level design alternatives		Type IV
	Select product component solutions		Type IV
	Prepare high-level design document		Type IV
	Review high-level design		Type IV
	Develop detailed design	Develop design model	Type IV
		Design Database	Type IV
		Prepare software design description document	Type III
		Review detailed design	Type IV
Implement the design	Implement product components		Type IV
	5.2.2.Perform unit tests		Type IV
	5.2.3.Peer review source codes		Type IV
	5.2.4.Develop product support documentation		Type II
<b>VV</b>	<b>Verification and Validation</b>		
5.1.Planning verification and validation activities	5.1.1.Planning peer reviews		Type IV

Figure 58 Process list (CASE STUDY I) - Part 1



	5.1.2.Planning test activities	5.1.2.1.Planning integration tests	Type IV
		5.1.2.2.Planning qualification tests	Type IV
		5.1.2.3.Planning acceptance tests	Type IV
5.2.Establish the verification and validation environment	5.1.3.Planning prototyping		Type IV
5.3.Perform verification	5.3.1.Perform integration tests	5.3.1.1.Execute integration tests	Type II
		5.3.1.2.Analyze integration test results	Type IV
	5.3.2.Perform qualification tests	5.3.2.1.Execute qualification tests	Type II
		5.3.2.2.Analyze qualification test results	Type IV
	5.3.3.Perform peer reviews	5.3.3.1.Conduct peer reviews	Type IV
		5.3.3.2.Analyze peer review data	Type IV
5.4.Perform validation	5.4.1.Prototyping		Type IV
	5.4.2.Design and preparation of acceptance tests		Type IV
	5.4.3.Perform acceptance tests	5.4.3.1.Execute acceptance tests	Type 0
		5.4.3.2.Analyze acceptance test results	Type IV
<b>PcM</b>	<b>Process Management Process</b>	<b>Skipped</b>	
<b>MA</b>	<b>Measurement and Analysis</b>	<b>Skipped</b>	
<b>SQA</b>	<b>Software Quality Assurance Process</b>	<b>Skipped</b>	
<b>RM</b>	<b>Risk Management</b>	<b>Skipped</b>	
<b>OT</b>	<b>Organization Training</b>	<b>Skipped</b>	
<b>DAR</b>	<b>Decision Analysis and Resolution</b>	<b>Skipped</b>	
<b>OT</b>	<b>Organization Training</b>	<b>Skipped</b>	
<b>DAR</b>	<b>Decision Analysis and Resolution</b>	<b>Skipped</b>	

**Figure 59 Process list (CASE STUDY I) - Part 2**

## **APPENDIX D: PROCESS MAPPING (CASE STUDY II)**

Case Study II - Sequence Break-Up Form					
Step	Action	Tool	Pseudo Inputs	Pseudo Outputs	Pseudo Service
<b>KY-020-621</b>					
Seq1					
3.5	Send e-mail to stakeholders	email	recipientAddress, messageContent	statusCode	email.send
4	Create a corrective action task, add related docs and CRs	-	-	-	-
4.a	Create a related task	ChM	crID, crPath	taskID	ChM.createTask
4.b	Get related items	ChM	crID, crPath	itemID	ChM.getLinkedItems
4.c	Add related items	ChM	taskID, itemID	linkID	ChM.createLink
Seq2					
10	Send e-mail to PM, CI Responsible, CM and responsible	email	recipientAddress, messageContent	statusCode	email.send
11	Provide access to CR responsible for related files	SCM	userID, filePath	statusCode	SCM.createFileAccessRights
Seq3					
15	Assign new version number if CCB asked for document distribution. Distribute document	-	-	-	-
15.a	Create label with new version number	SCM	labelName, labelPath	labelID	SCM.createLabel
15.b	Send e-mail to stakeholders	email	recipientAddress, messageContent	statusCode	email.send
<b>UG-010-84</b>					
Seq4					
2	Create SRD wrt template	-	-	-	-
2.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion
2.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument
<b>UG-040-83</b>					
Seq5					
1	Generate Software Requirements Document wrt template in "EK-1"	-	-	-	-
1.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion
1.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument
3	Generate Interface Requirements Document wrt template in "EK-1"	-	-	-	-
<b>UG-070-82</b>					
Seq6					
6	Store all documents and information related to unit test in CM tool	SCM	filePath, fileName	statusCode	SCM.updateFile
7	Assign a label to all units where coding is complete before unit tests	SCM	labelName, labelPath	labelID	SCM.createLabel
<b>UG-070-83</b>					
Seq7					
2	Execute tests and fill a test report form for the CI wrt "KG-100 Test Records"	-	-	-	-
2.a	Execute tests	TM	testID	executionID	TM.executeTest
2.b	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion
2.c	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument
2.2	Append test results or their location in CM tool to Test Report Form	file	filePath	statusCode	file.appendFile
3	Store test outputs, results and other information in CM tool	SCM	filePath, fileName	statusCode	SCM.updateFile
<b>UG-070-86</b>					
Seq8					
3	Continue unit testing until all units are integrated without problems	-	-	-	-
3.a	Check if all unit-tests are successful	TM	projectID	statusCode	TM.checkUnitTests
3.b	Execute tests	TM	testID	executionID	TM.executeTest
4	Store test outputs, results and other information in CM tool	SCM	filePath, fileName	statusCode	SCM.updateFile
<b>UG-070-87</b>					
Seq9					
5	Employ "Code Review Control List - EK-7"	TM	projectID	statusCode	TM.performCodeReview
6	Fill "Review Preparation Form"	-	-	-	-
6.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion
6.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument
<b>UG-070-89</b>					
Seq10					
1	Inform PM and CM when unit integration and tests are complete and CI is ready for the next level of tests	email	recipientAddress, messageContent	statusCode	email.send
2	Assign version number 1.00 to CI and its units	SCM	labelName, labelPath	labelID	SCM.createLabel

**Figure 60 Process Mapping for Case Study II – Part 1**

UG-190-82						
Seq11						
3	Label respective version of the document for the technical review	SCM	labelName, labelPath	labelID	SCM.createLabel	
4	Inform attendees on the date and location of technical review meeting	email	recipientAddress, messageContent	statusCode	email.send	
5	Provide access for attendees to the respective version of the document	SCM	userID, filePath	statusCode	SCM.createFileAccessRights	
UG-190-89						
Seq12						
4	Prepare "SCI Test Definitions Doc" wrt to the template in "EK-5".					
4.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
4.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
Seq13						
Generate "SCI Product Attributes" including executable software, source code, design and support documents for preparation to delivery wrt to the template in "EK-7"						
7						
7.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
7.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
Seq14						
Generate "Software Version Definition Doc" wrt "KY-051 Software Version Definition Doc Preparation Guideline".						
11						
11.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
11.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
Seq15						
20	Generate "SCI Test Report" wrt the template in "EK-9"					
20.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
20.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
UG-190-810						
Seq16						
Create HCI Test Definitions Document wrt the template in "EK-13".						
9						
9.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
9.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
Seq17						
Generate Initial Qualification Test Report wrt the template in "EK-15".						
19						
19.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
19.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
Seq18						
28	Generate HCI Test Report wrt the template in "EK-15"					
28.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
28.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
UG-190-811						
Seq19						
Generate System Integration and Test Plan as part of VV Plan in "EK-1"						
4						
4.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
4.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
Seq20						
Generate System Integration and Test Definitions Document wrt the template in "EK-19"						
8						
8.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
8.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
UG-190-812						
Seq21						
Generate System Test Plan as part of VV Plan in "EK-1"						
4						
4.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
4.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
Seq22						
Generate System Test Definitions Document wrt the template in "EK-23".						
7						
7.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	
7.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument	
Seq23						
Generate Internal System Test Report wrt the template in "EK-25".						
15						
15.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion	

**Figure 61 Process Mapping for Case Study II - Part 2**

15.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument
UG-190-813					
Seq24					
10	Generate System Acceptance Test Report wrt the template in "EK-25".				
10.a	Get template	SCM	filePath, fileName	statusCode	SCM.getLatestVersion
10.b	Generate document	DocGen	iniFile	statusCode	DocGen.generateDocument

**Figure 62 Process Mapping for Case Study II - Part 2**

## **APPENDIX E: PROCESS MODELS (CASE STUDY I)**

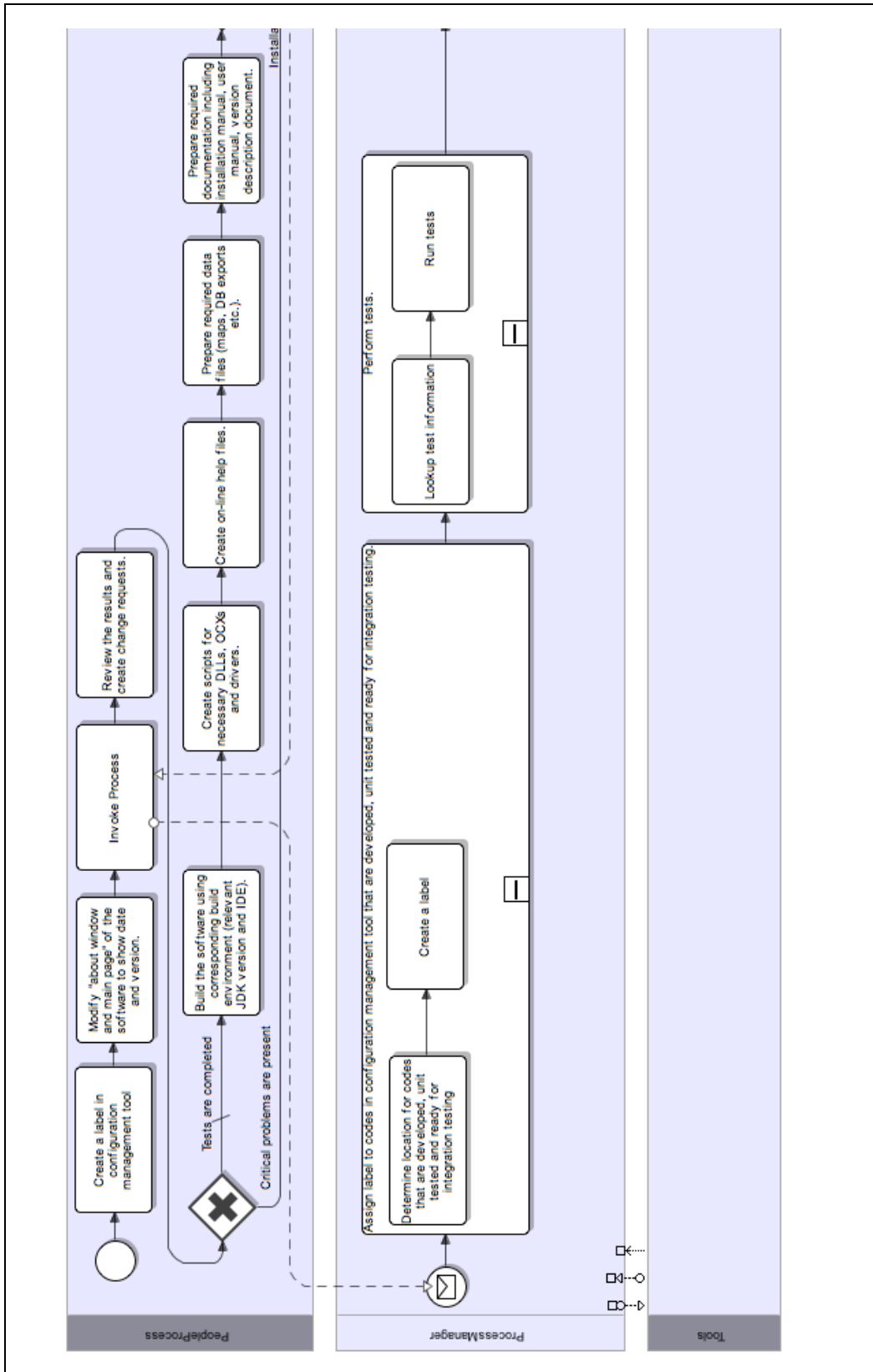


Figure 63 Process Model for BRPG2 - Part 1

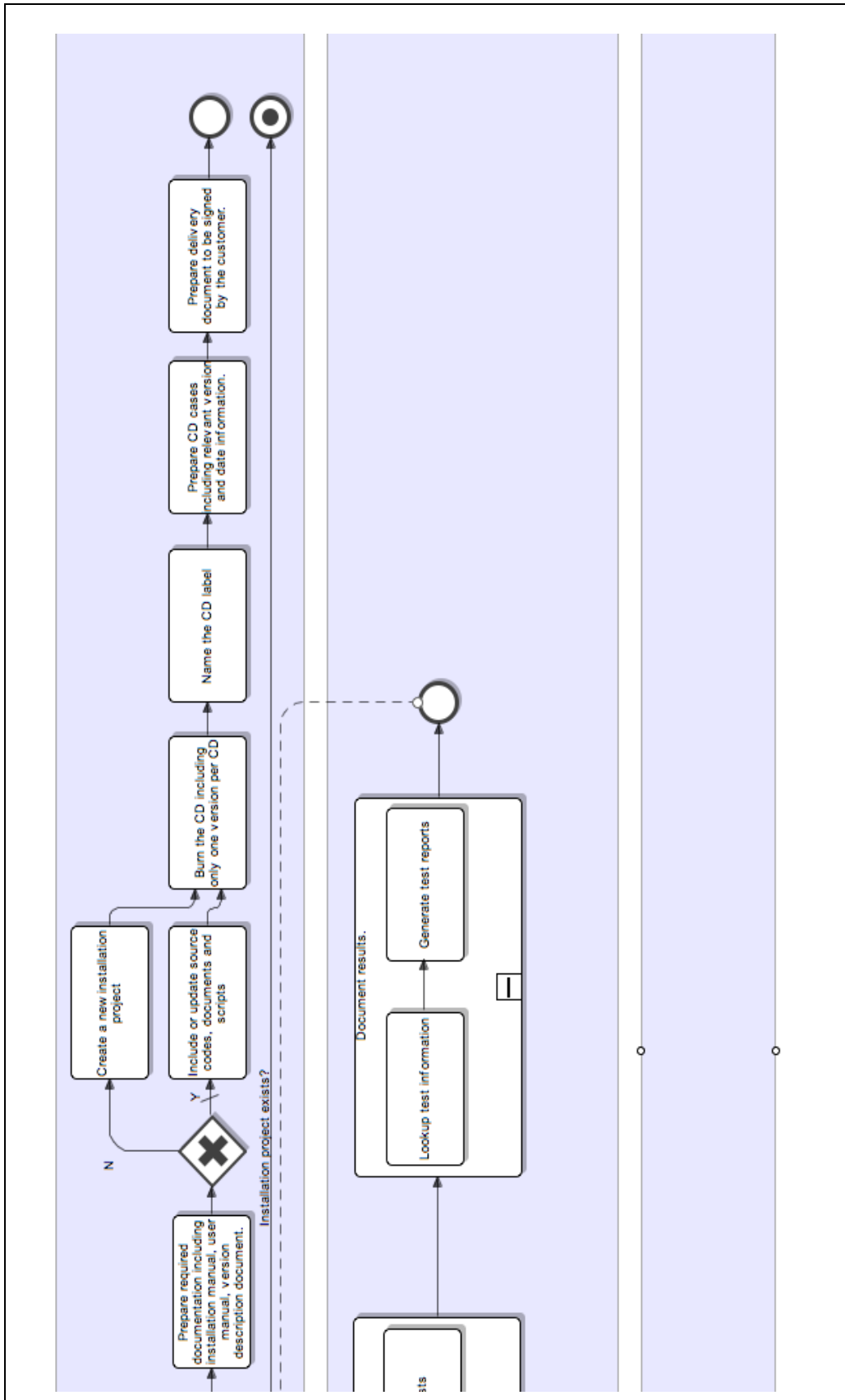


Figure 64 Process Model for BRPG2 - Part 2



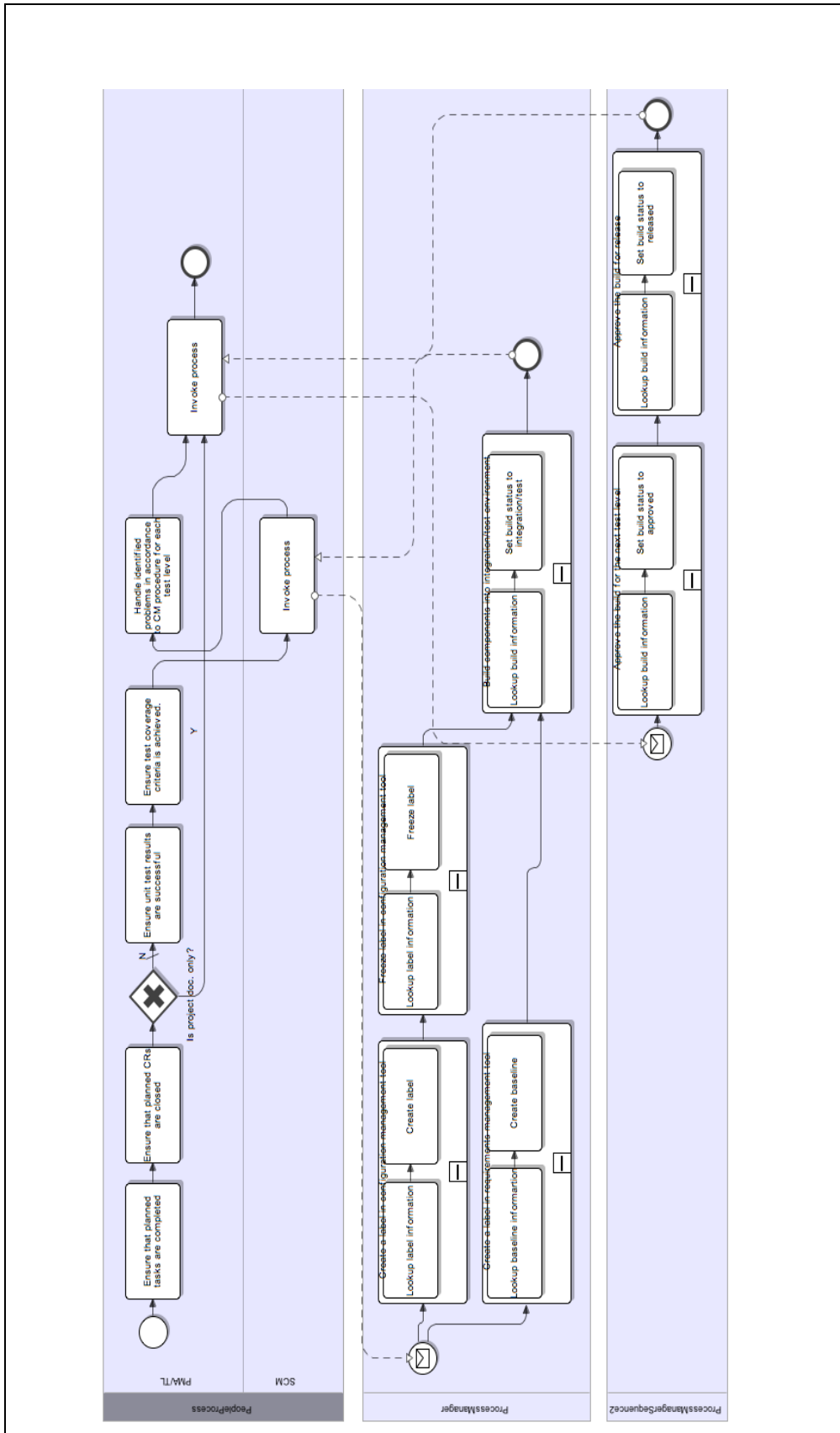


Figure 65 Process Model for CMG510

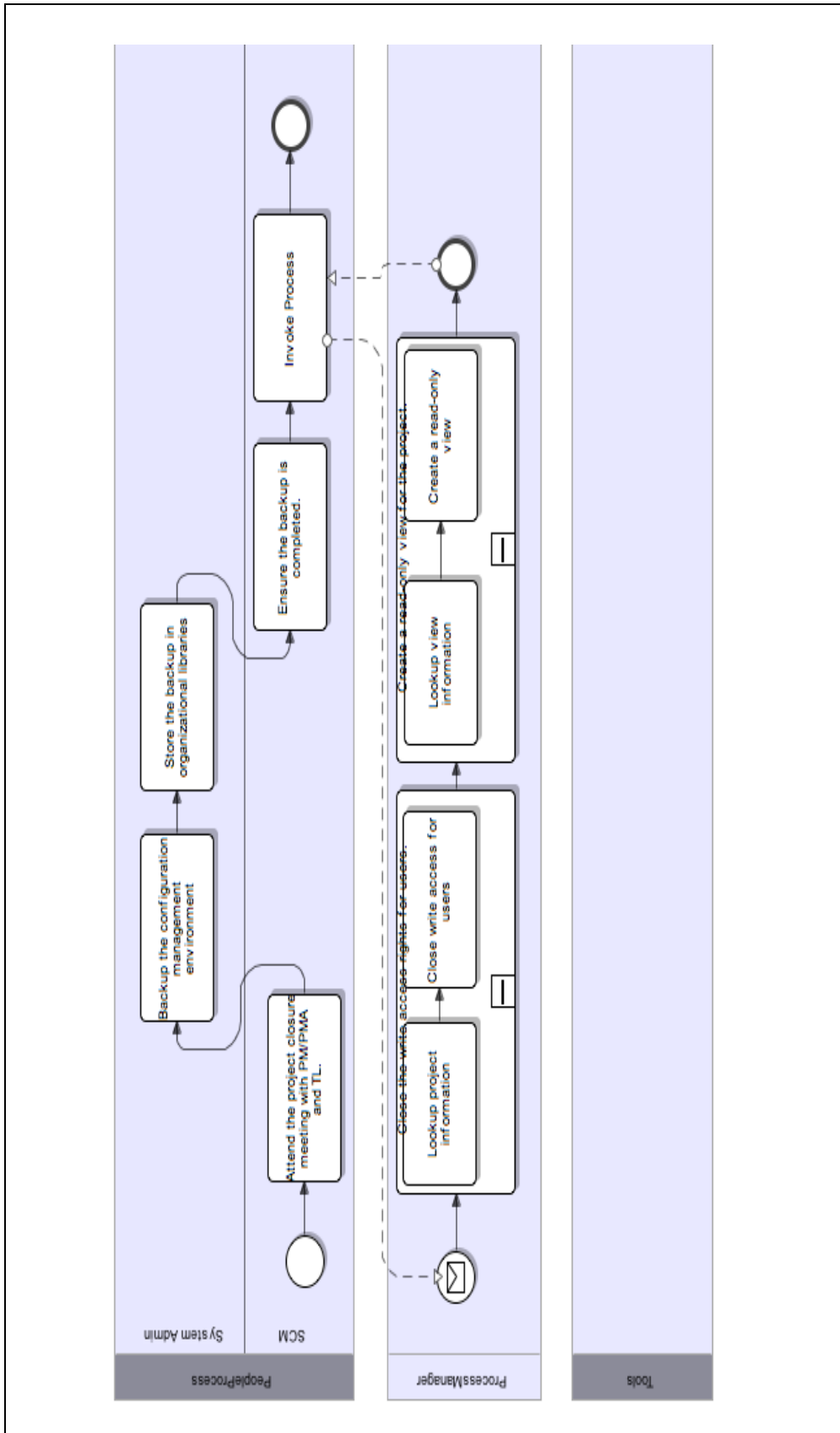


Figure 66 Process Model for CM513

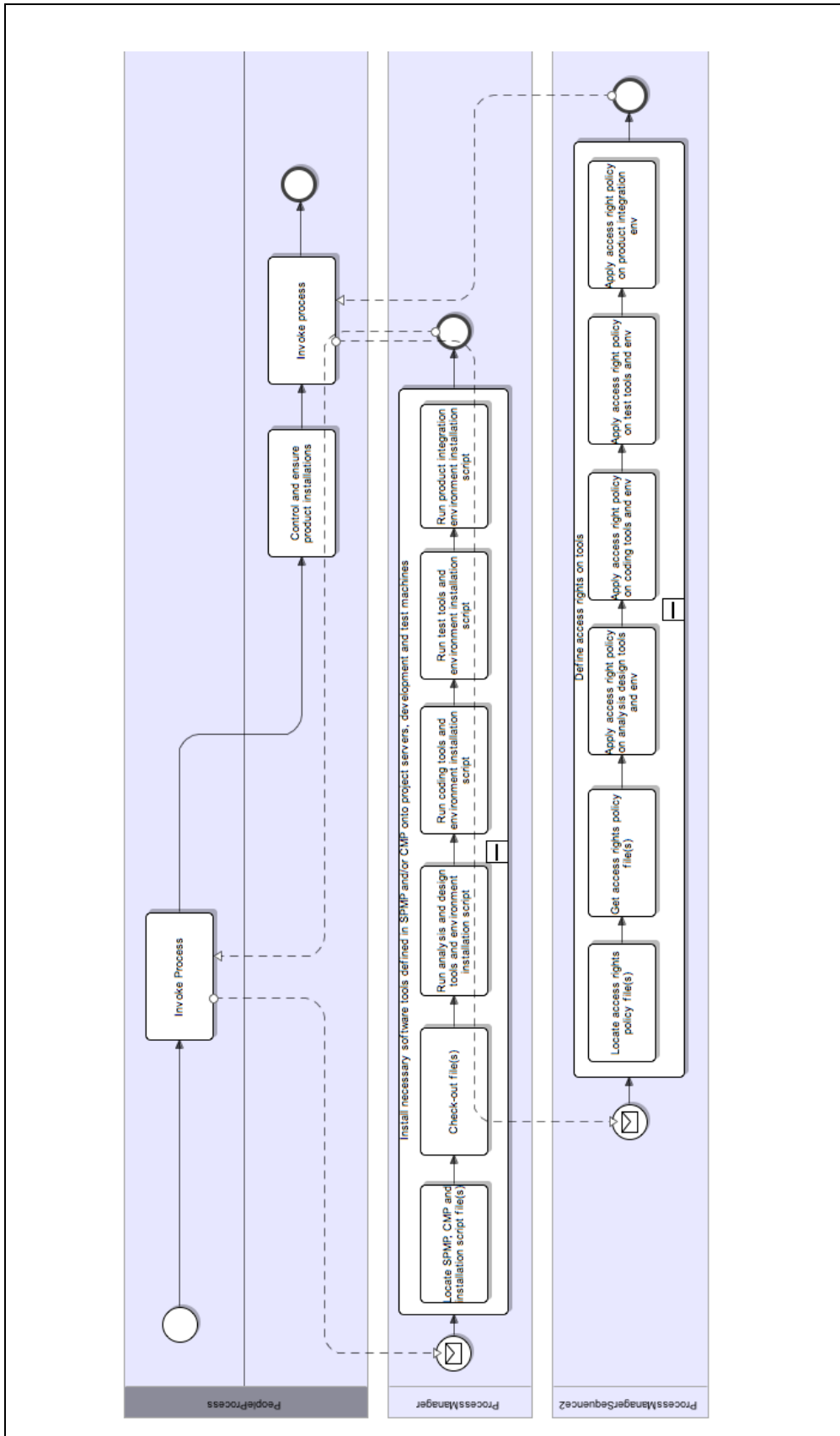


Figure 67 Process Model for CM533

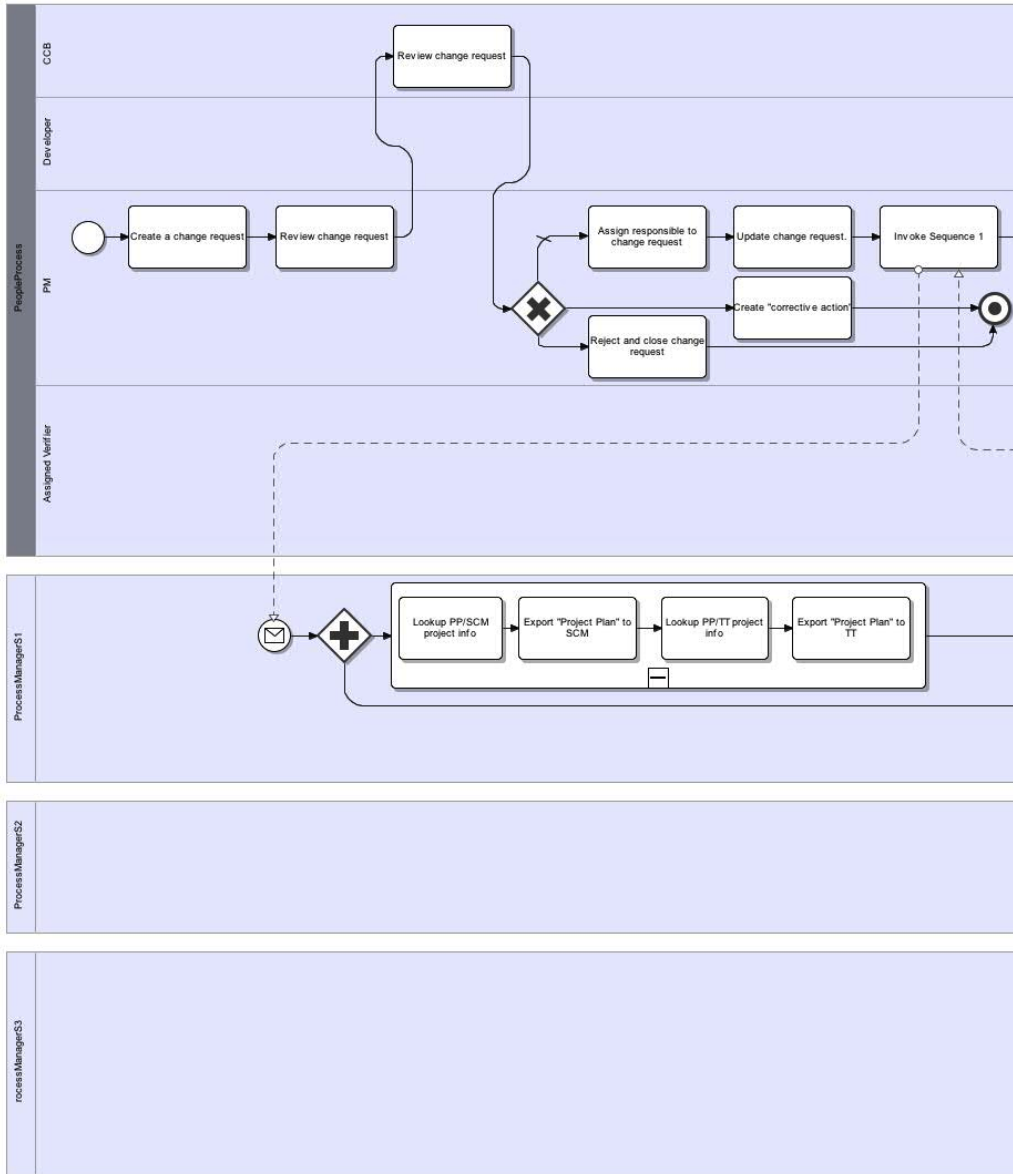


Figure 68 Process Model for CMG21-Part 1

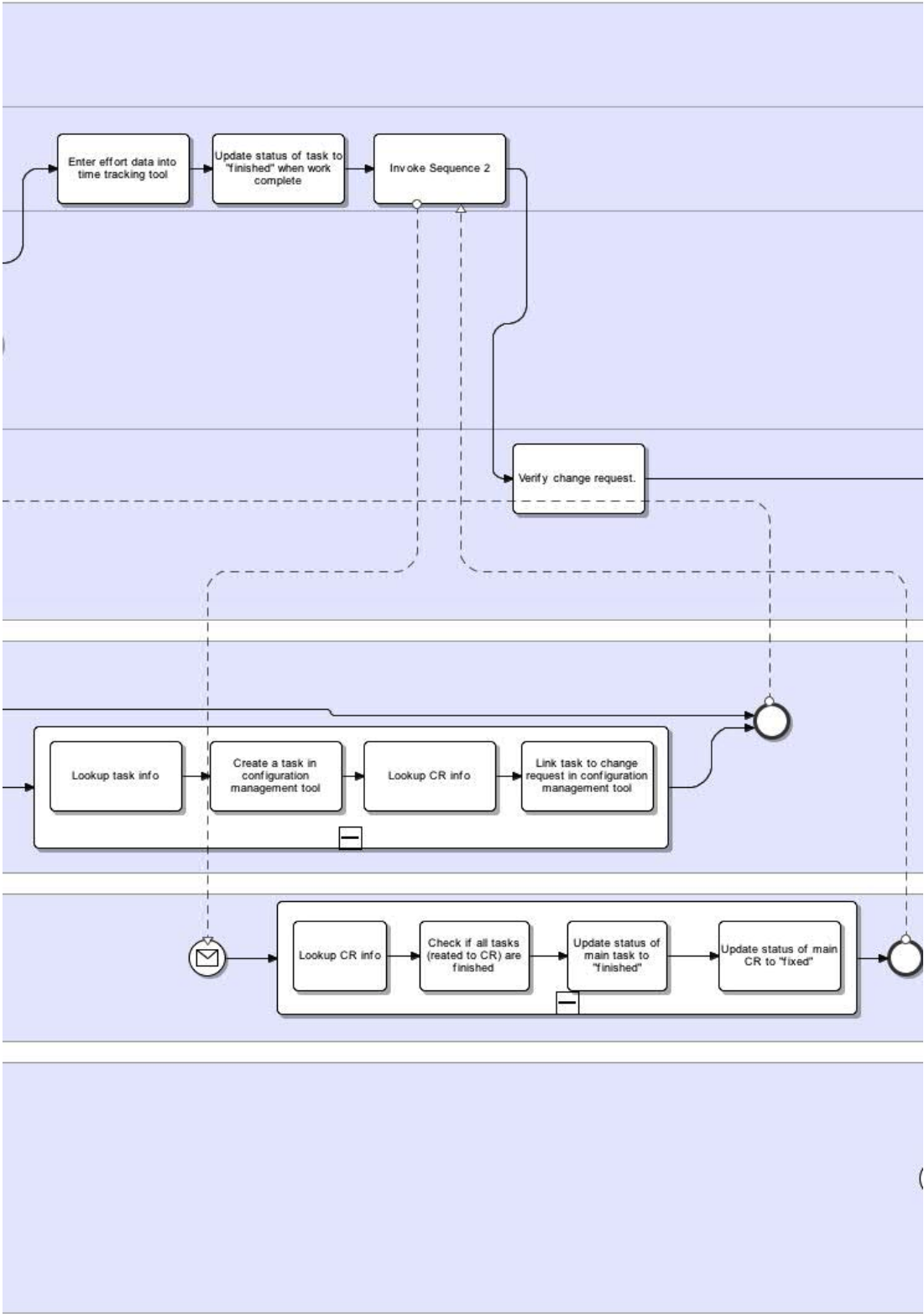


Figure 69 Process Model for CG21-Part 2

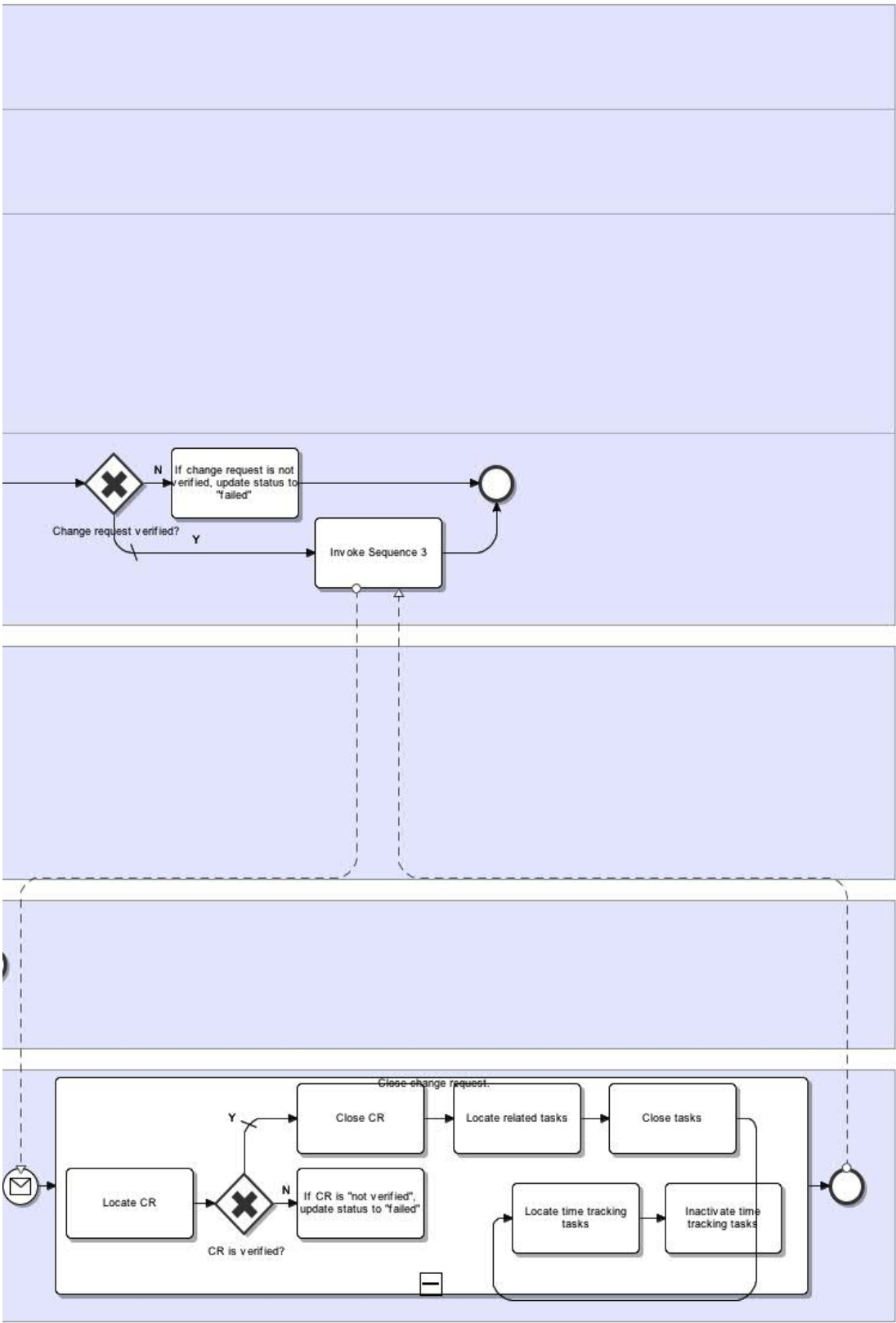


Figure 70 Process Model for CMG21-Part 3

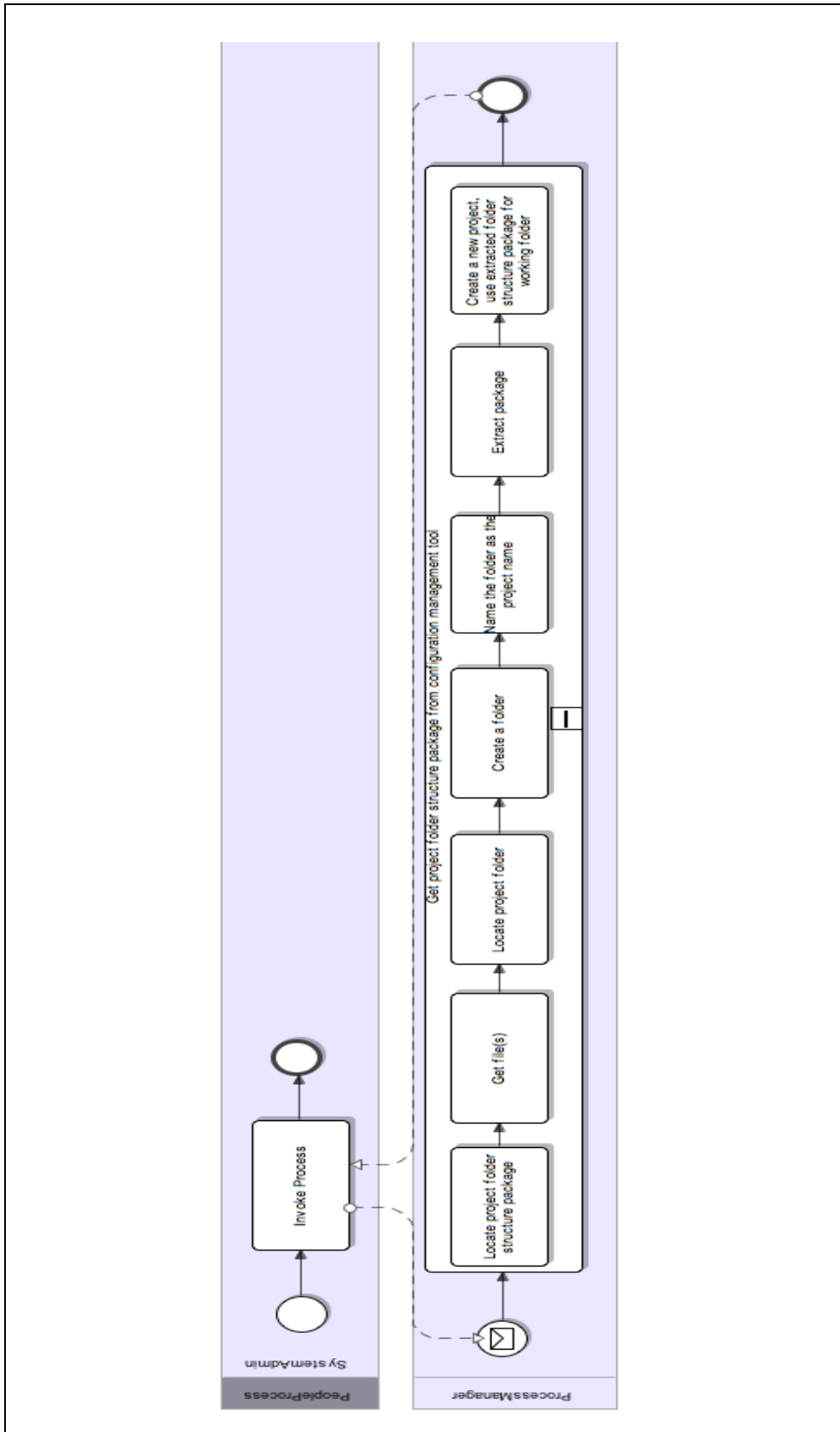


Figure 71 Process Model for CMTG211

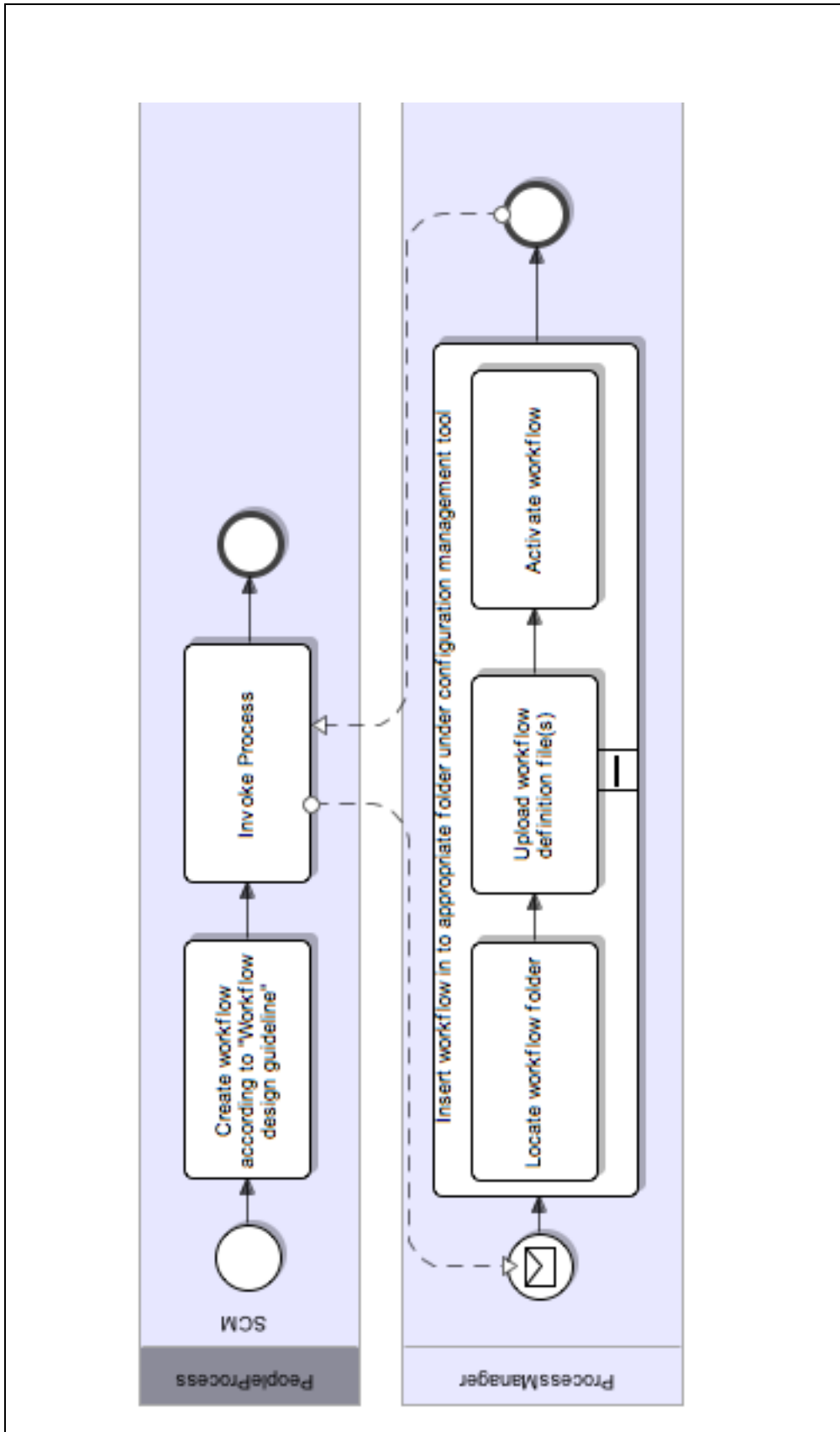


Figure 72 Process Model for CMTG212



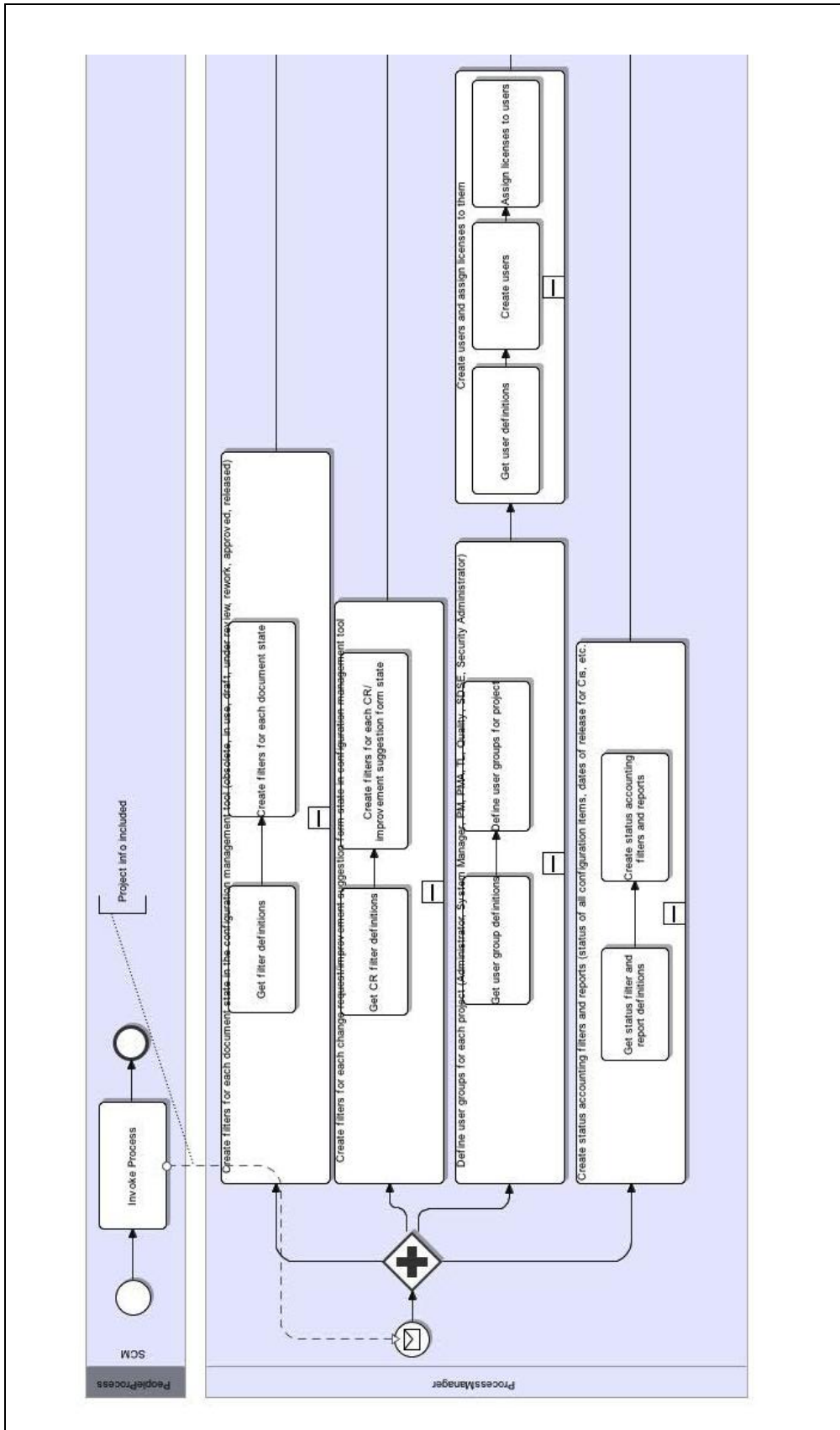


Figure 73 Process Model for CMTG213-216 Part 1

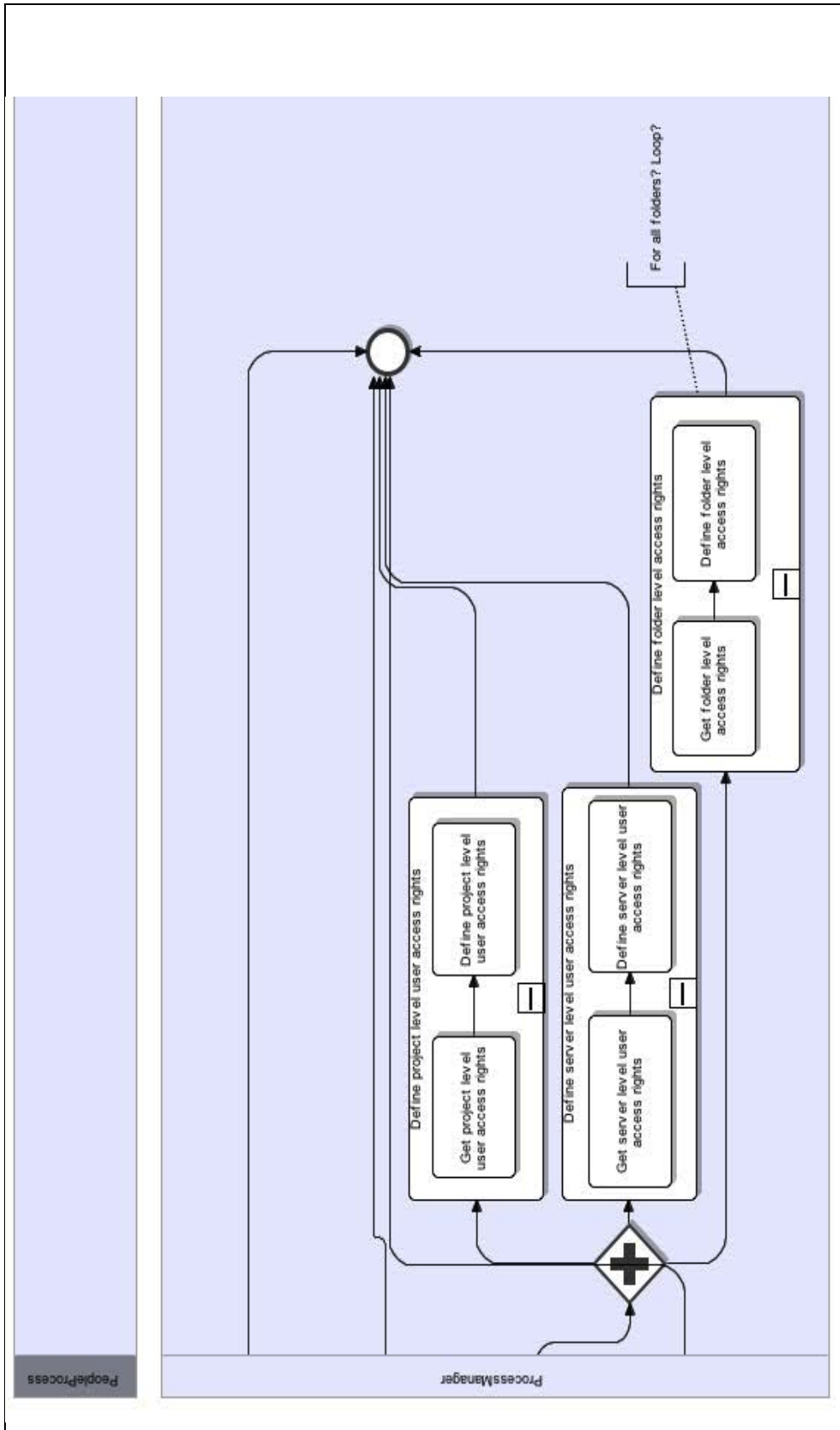


Figure 74 Process Model for CMTG21-216 Part2

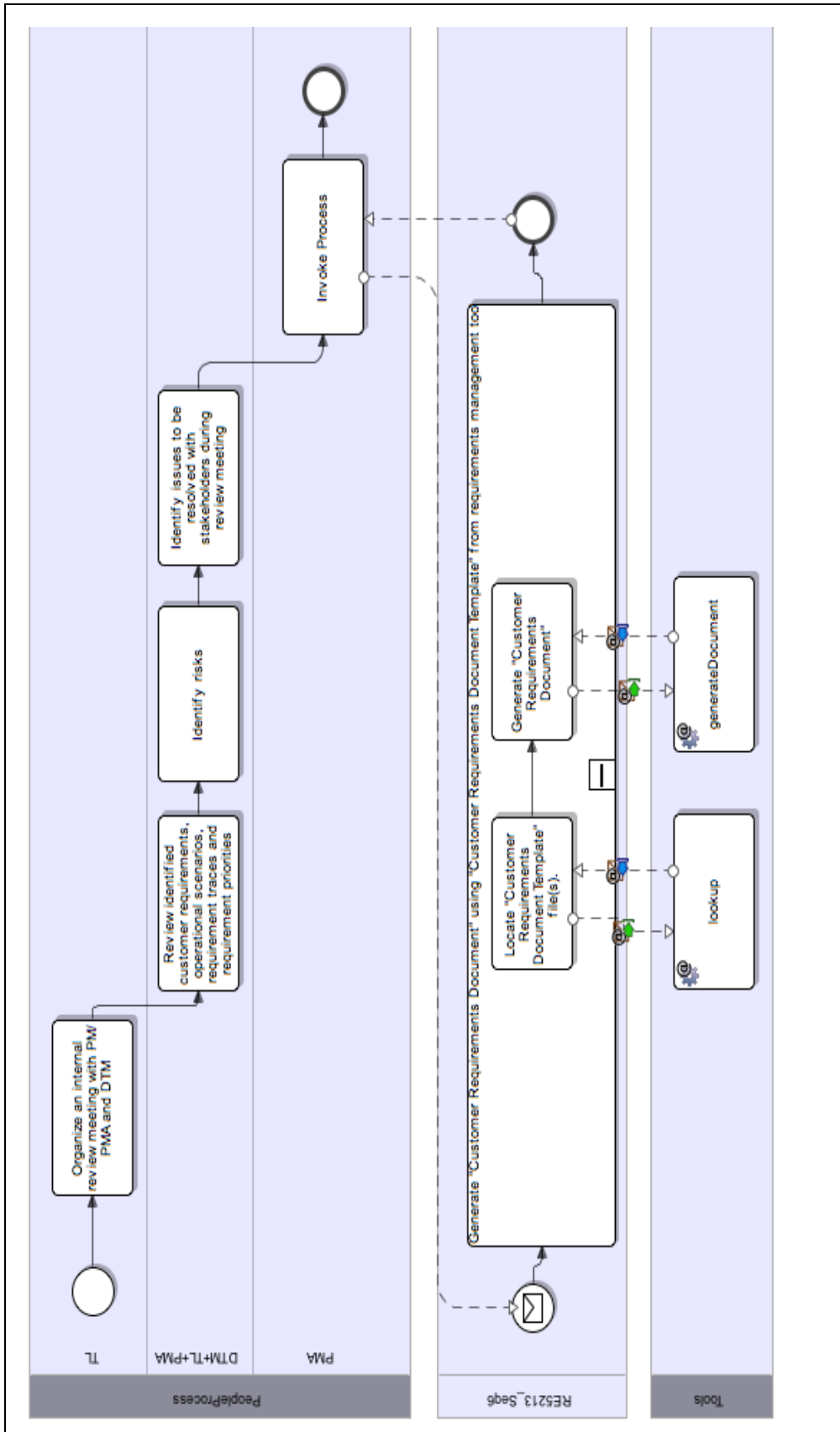


Figure 75 Process Model for RE5213

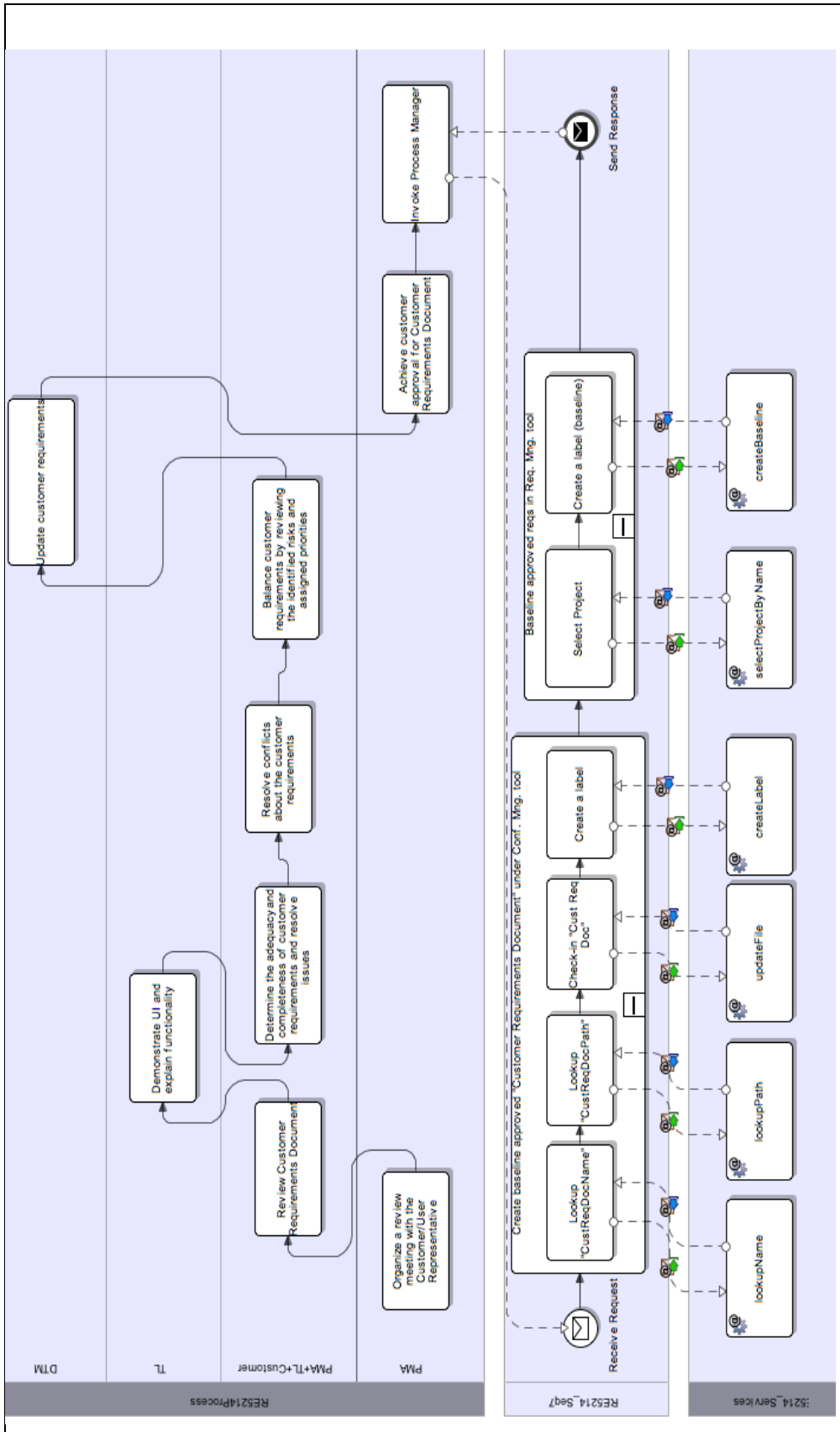


Figure 76 Process Model for RE5214

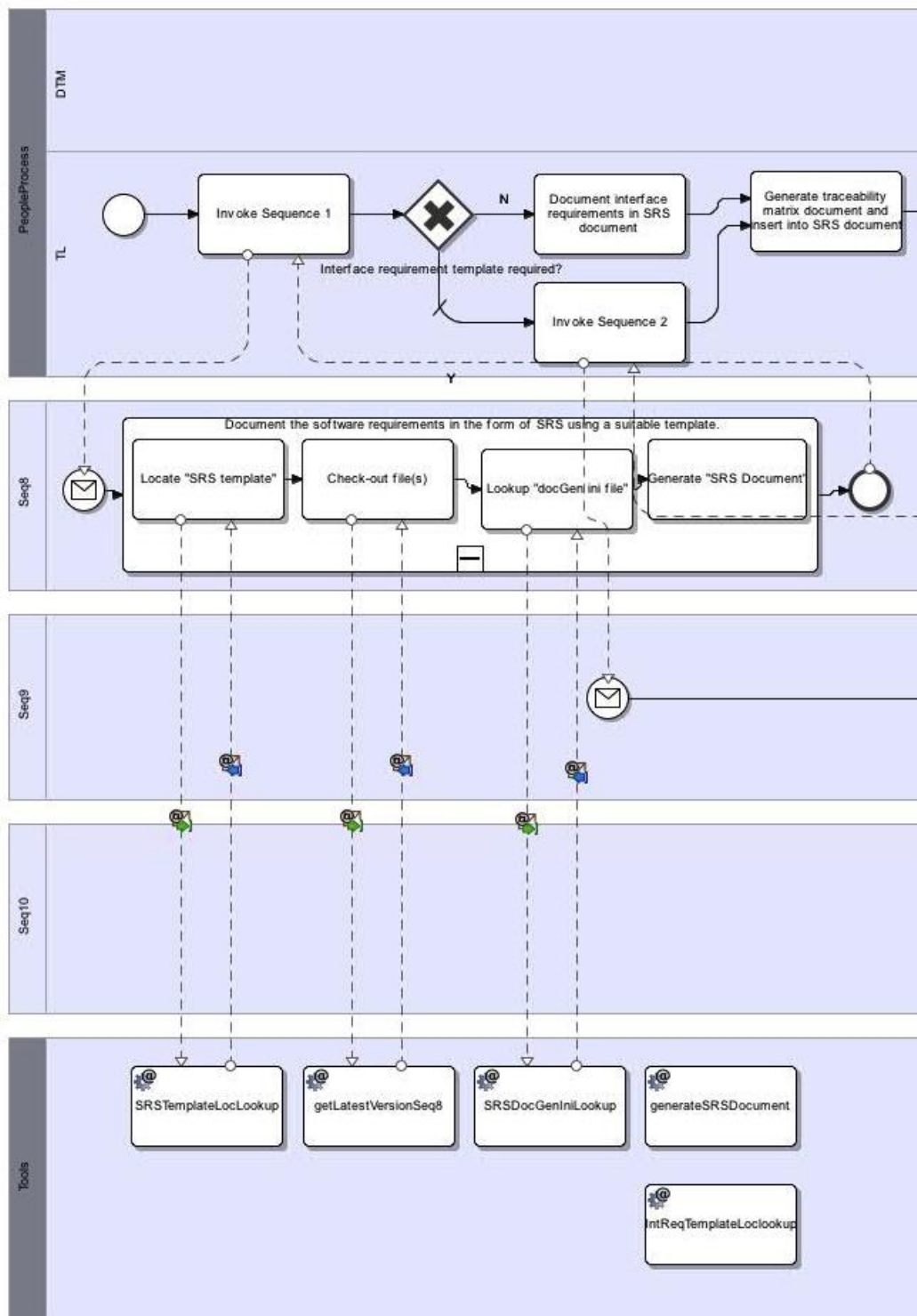


Figure 77 Process Model for RE52212 Part 1

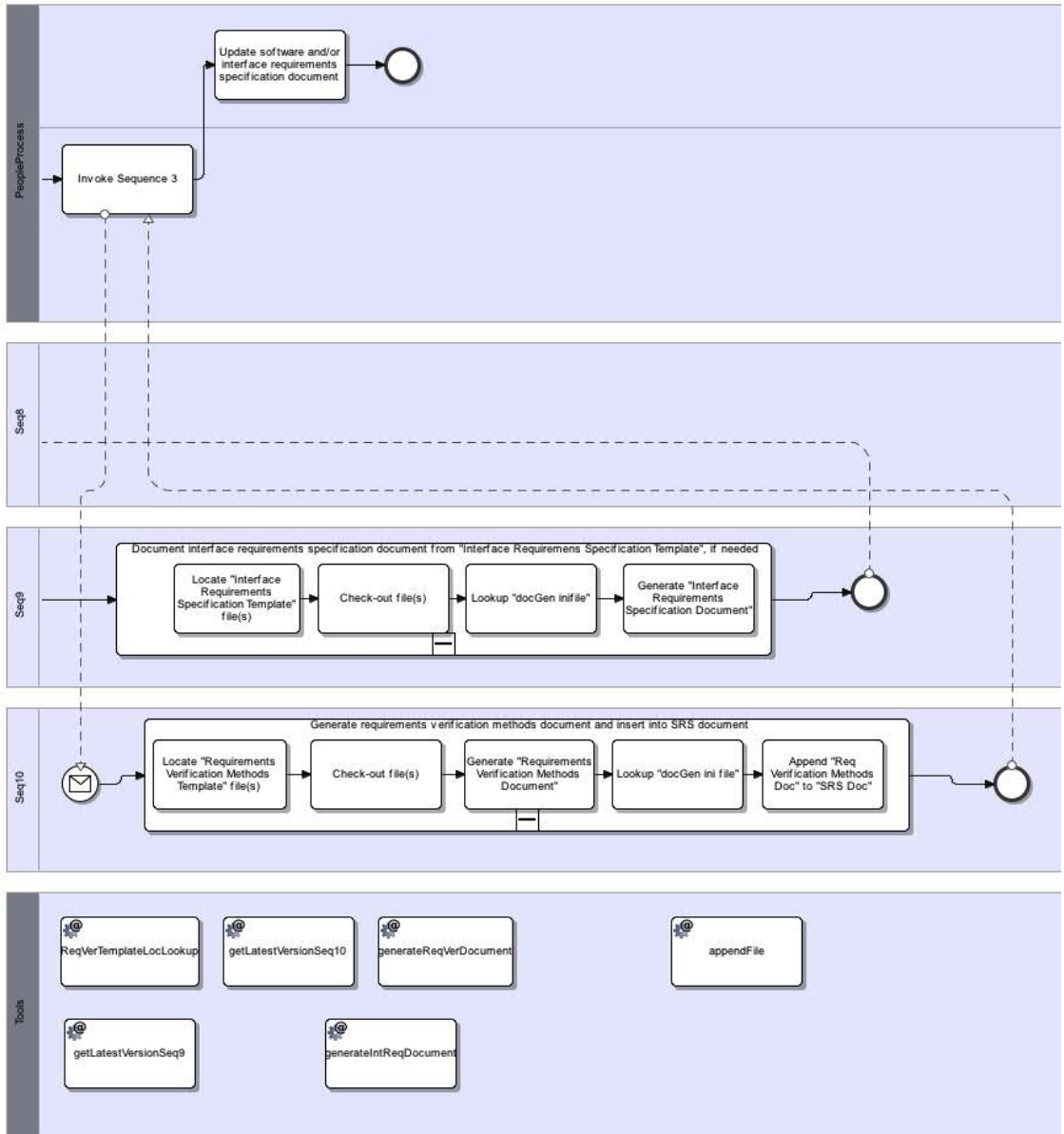


Figure 78 Process Model for RE52212 Part 2

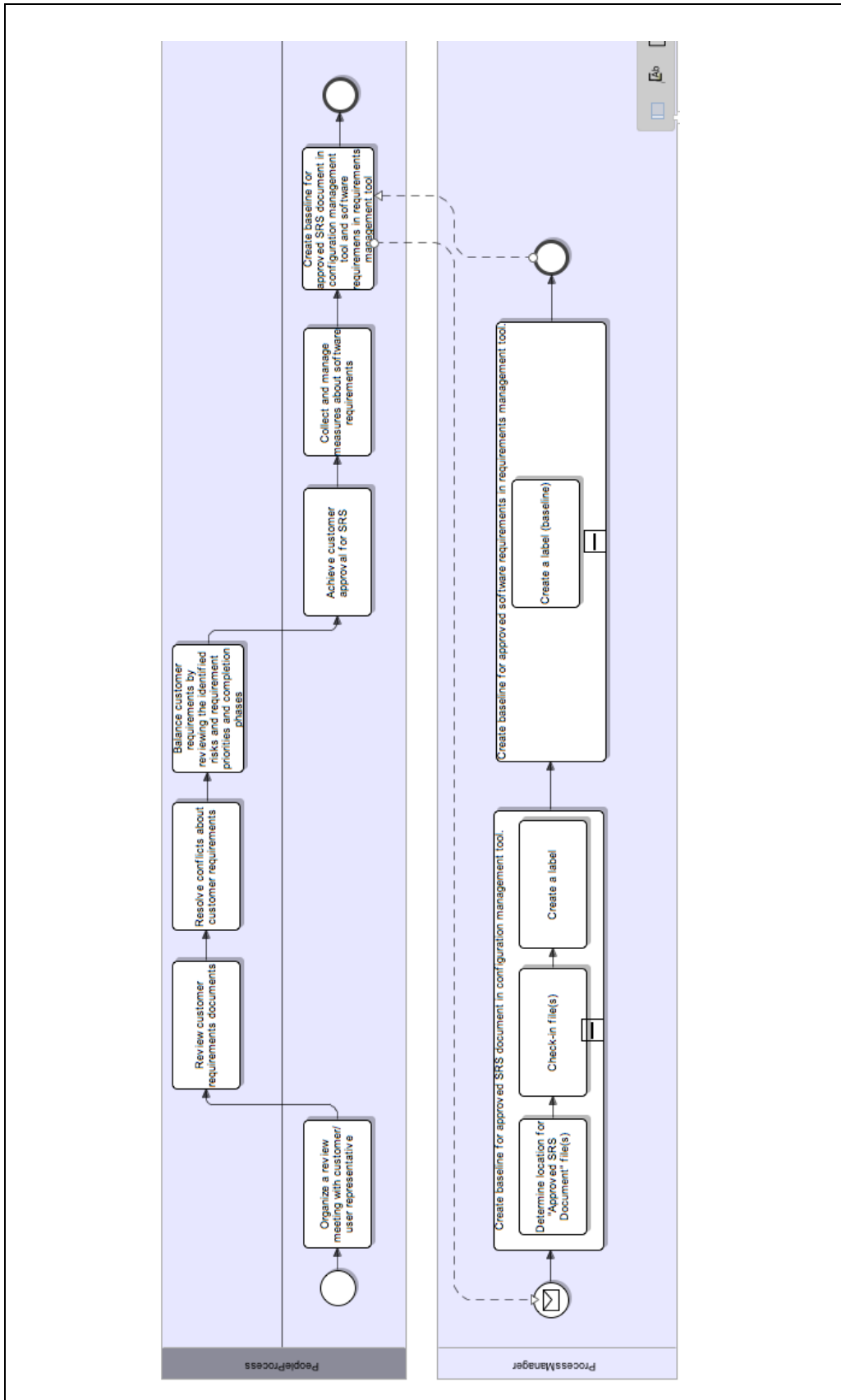


Figure 79 Process Model for RE5222

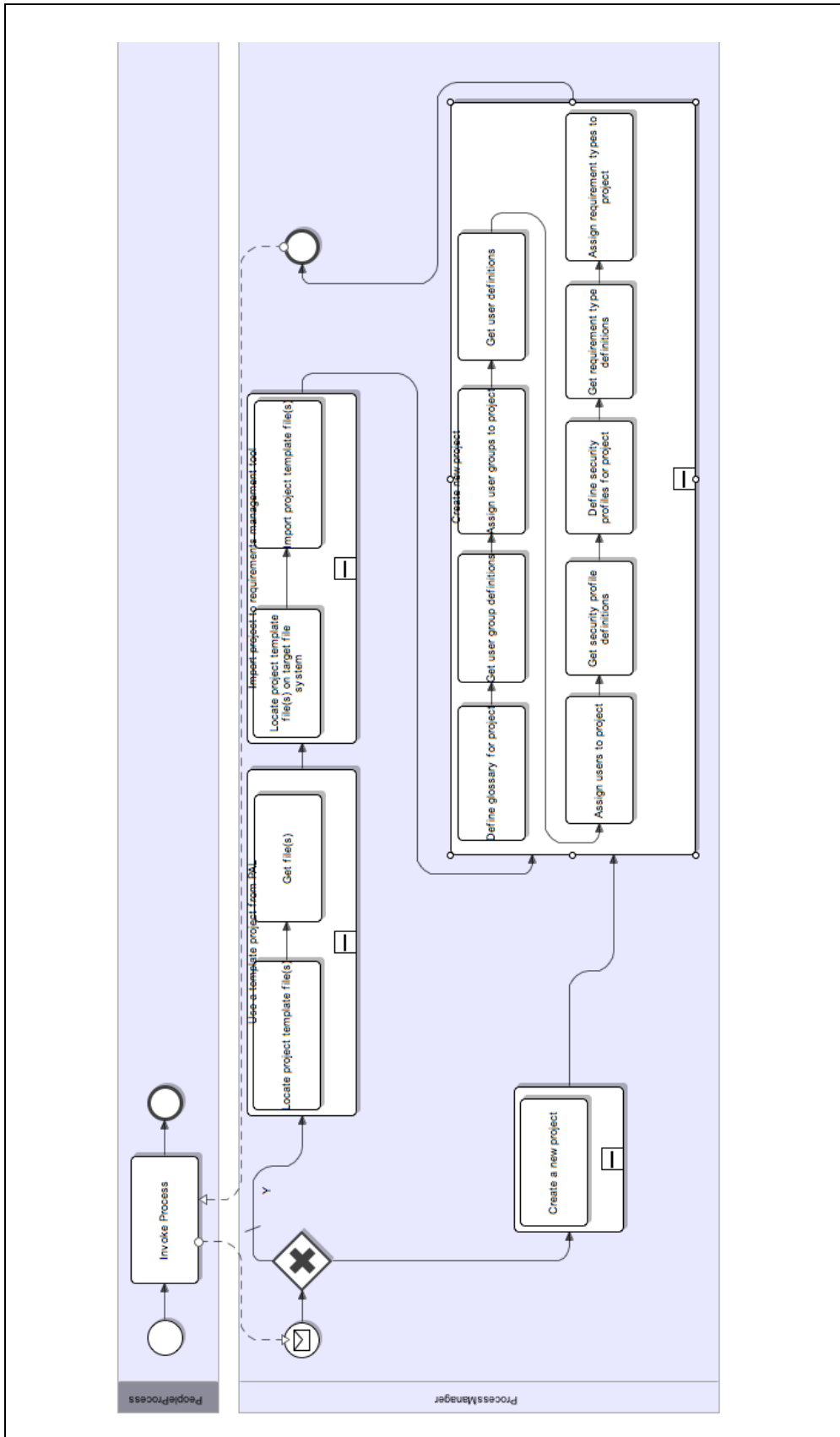


Figure 80 Process Model for RMTG21



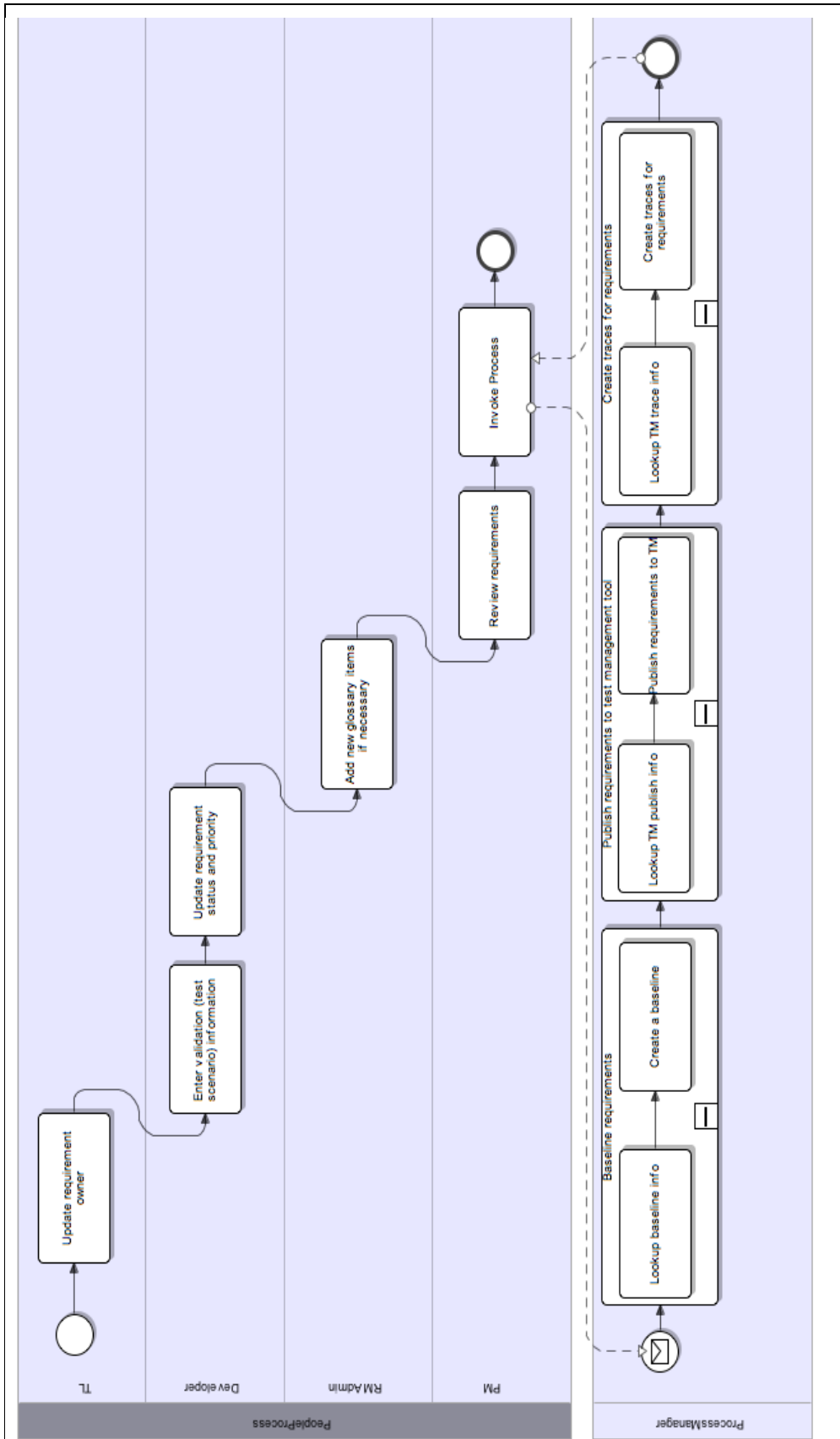


Figure 81 Process Model for RMTG22

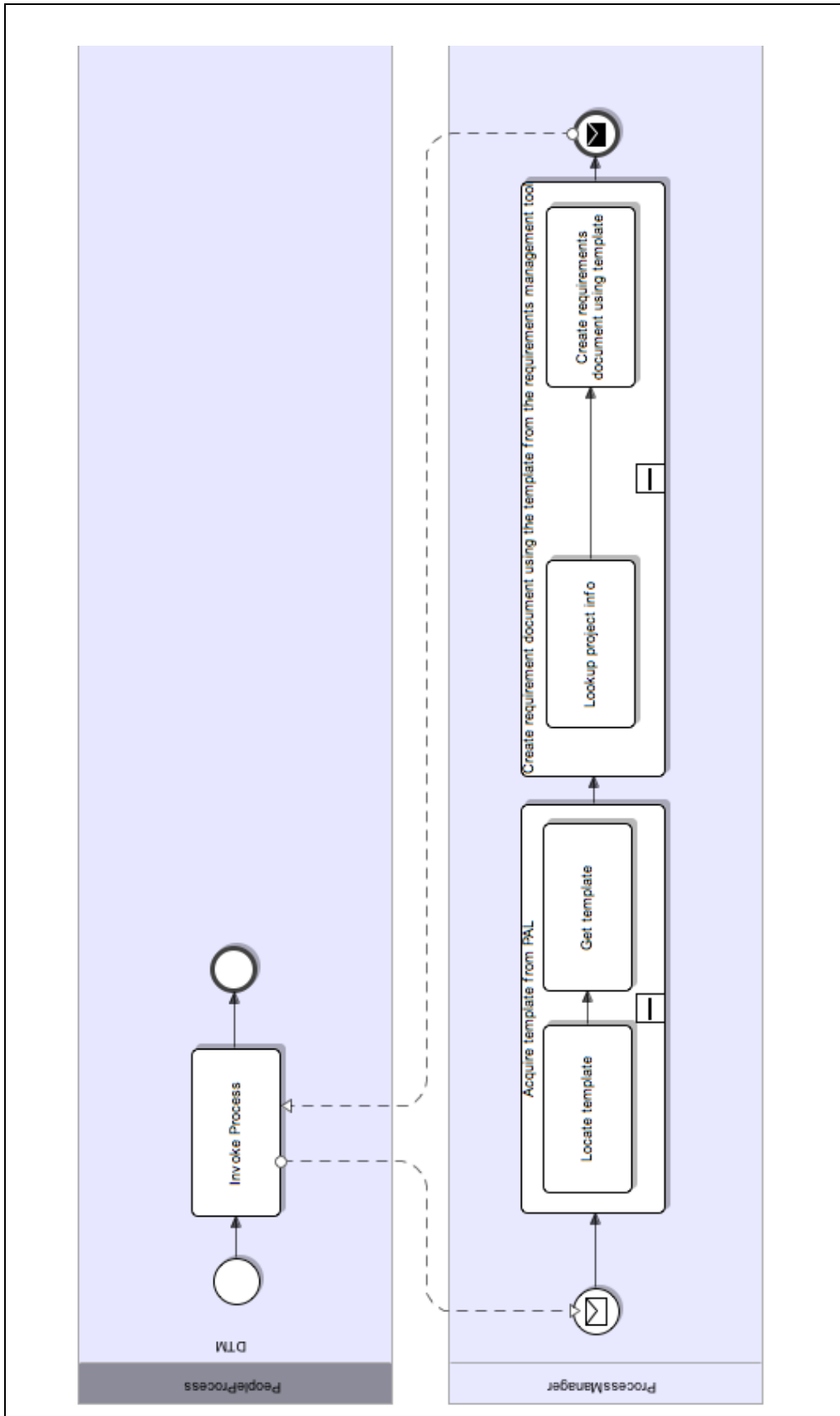


Figure 82 Process Model for RMT G23

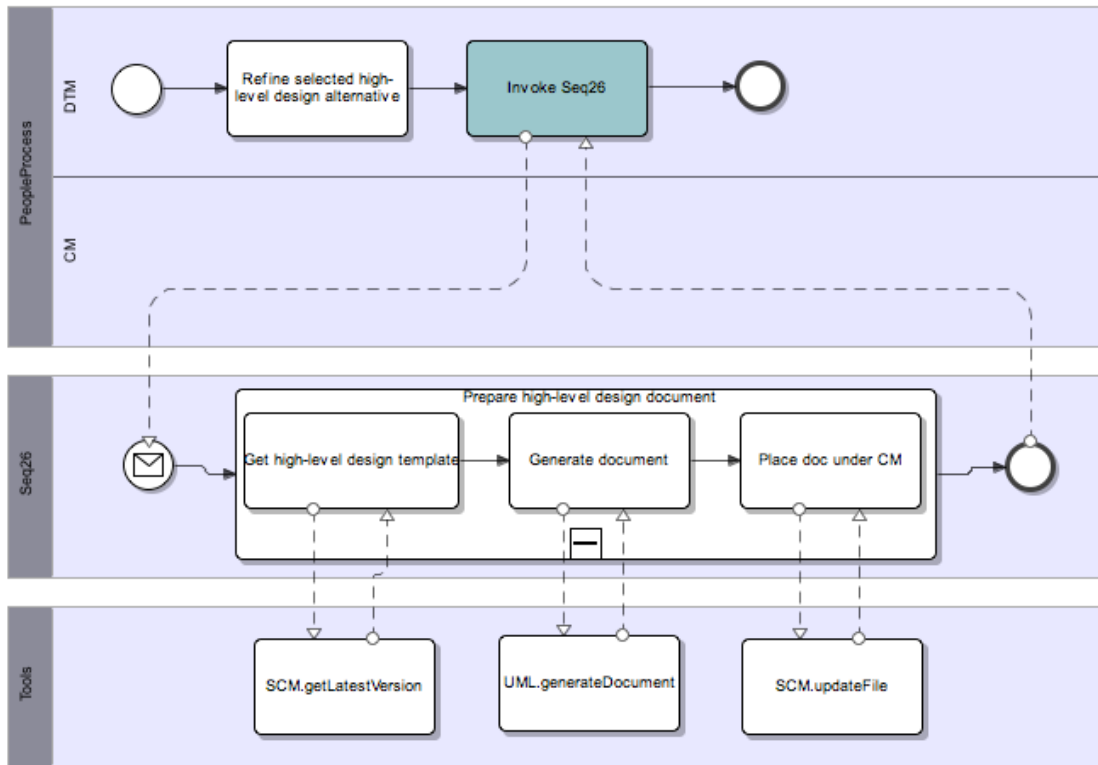


Figure 83 Process model for TS514

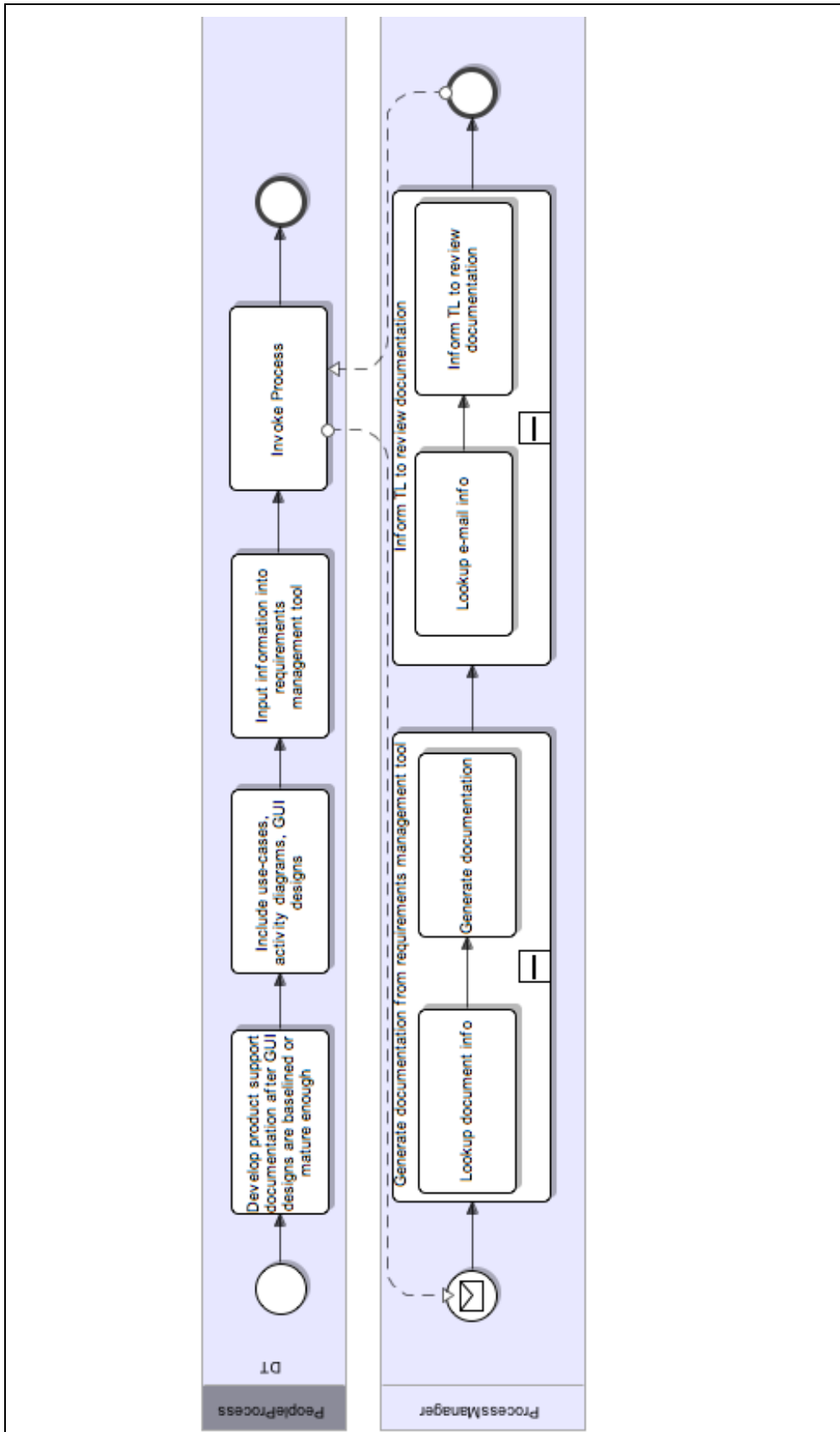


Figure 84 Process Model for TS524

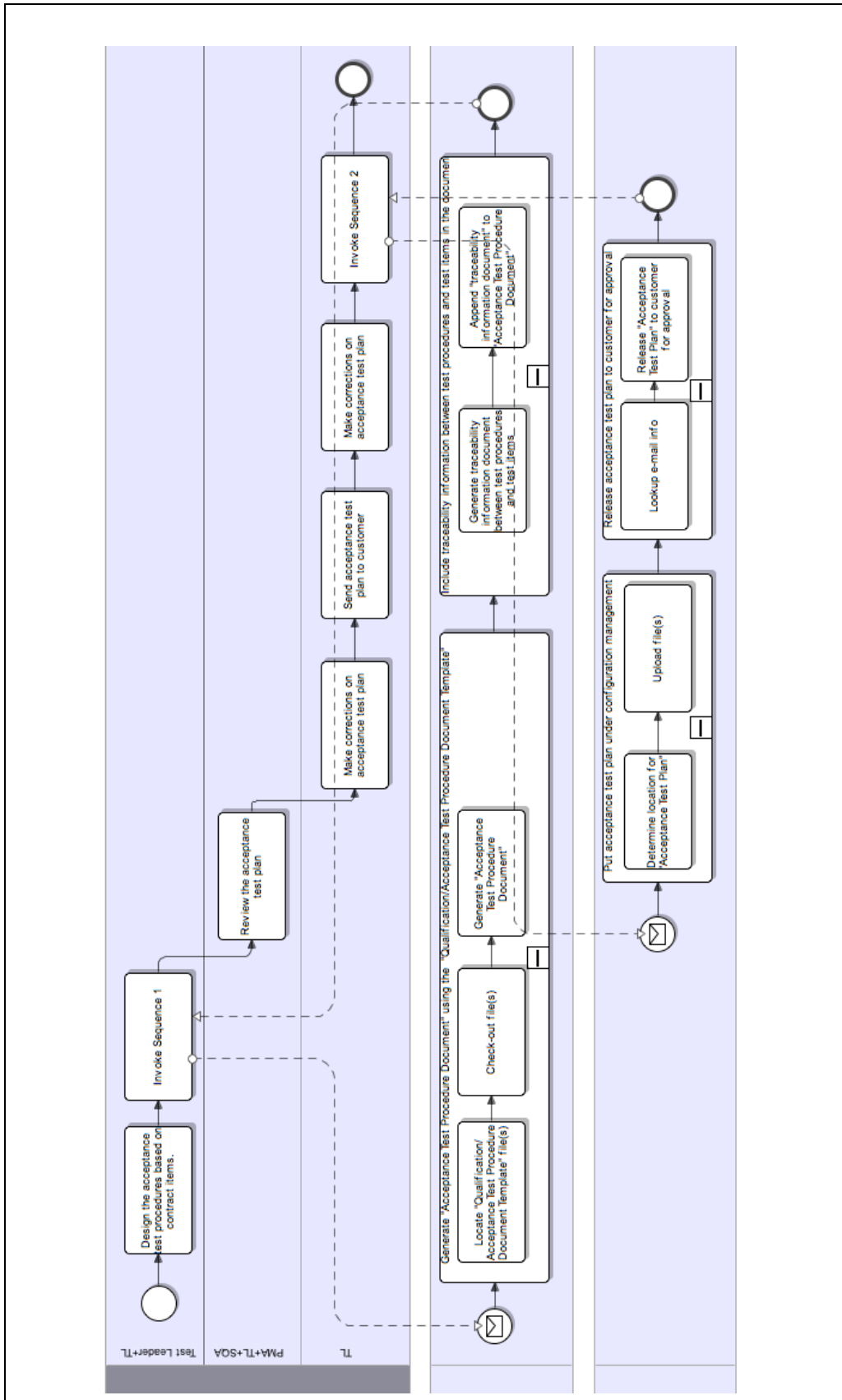


Figure 85 Process Model for VV542

## **APPENDIX F: PROCESS MODELS (CASE STUDY II)**

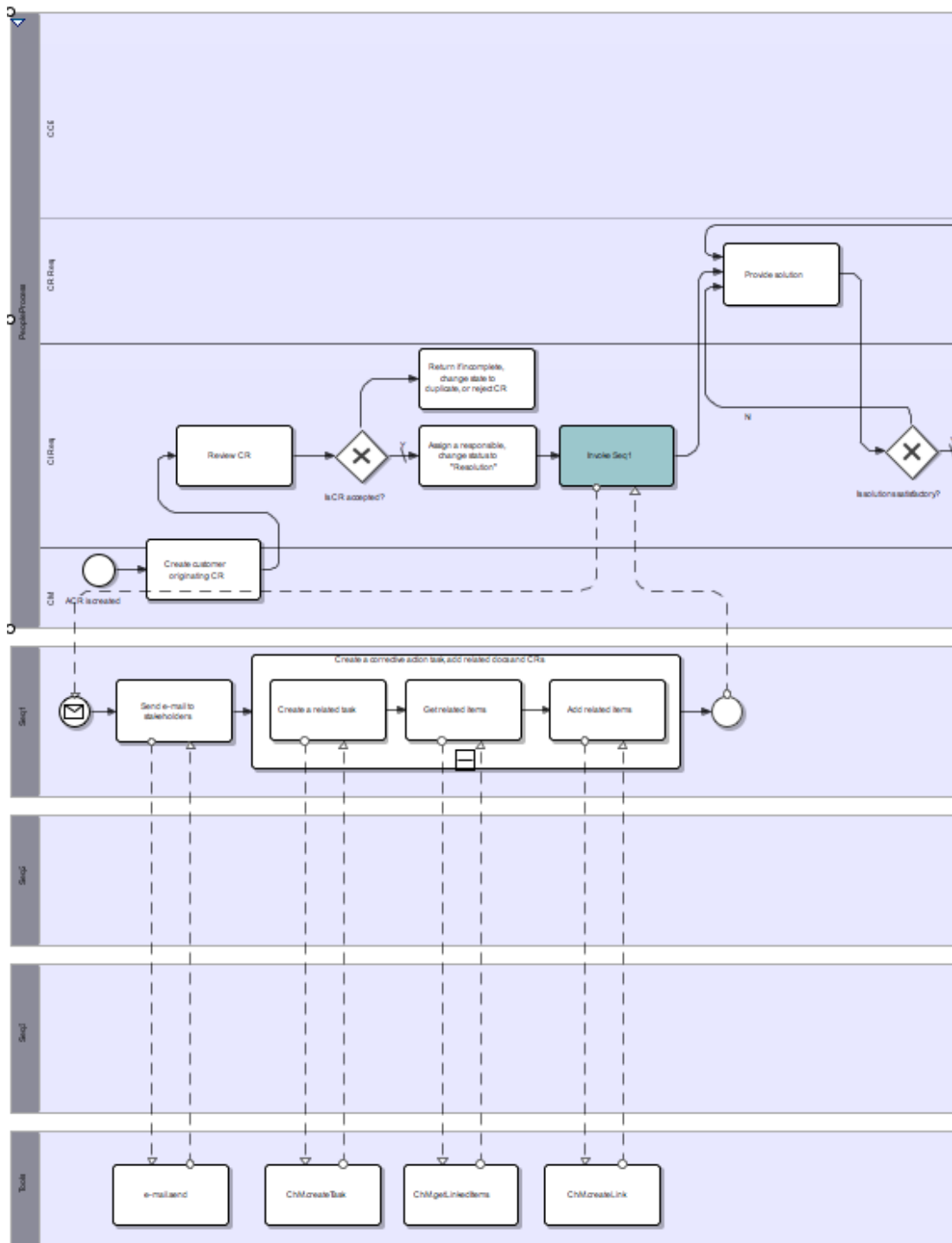
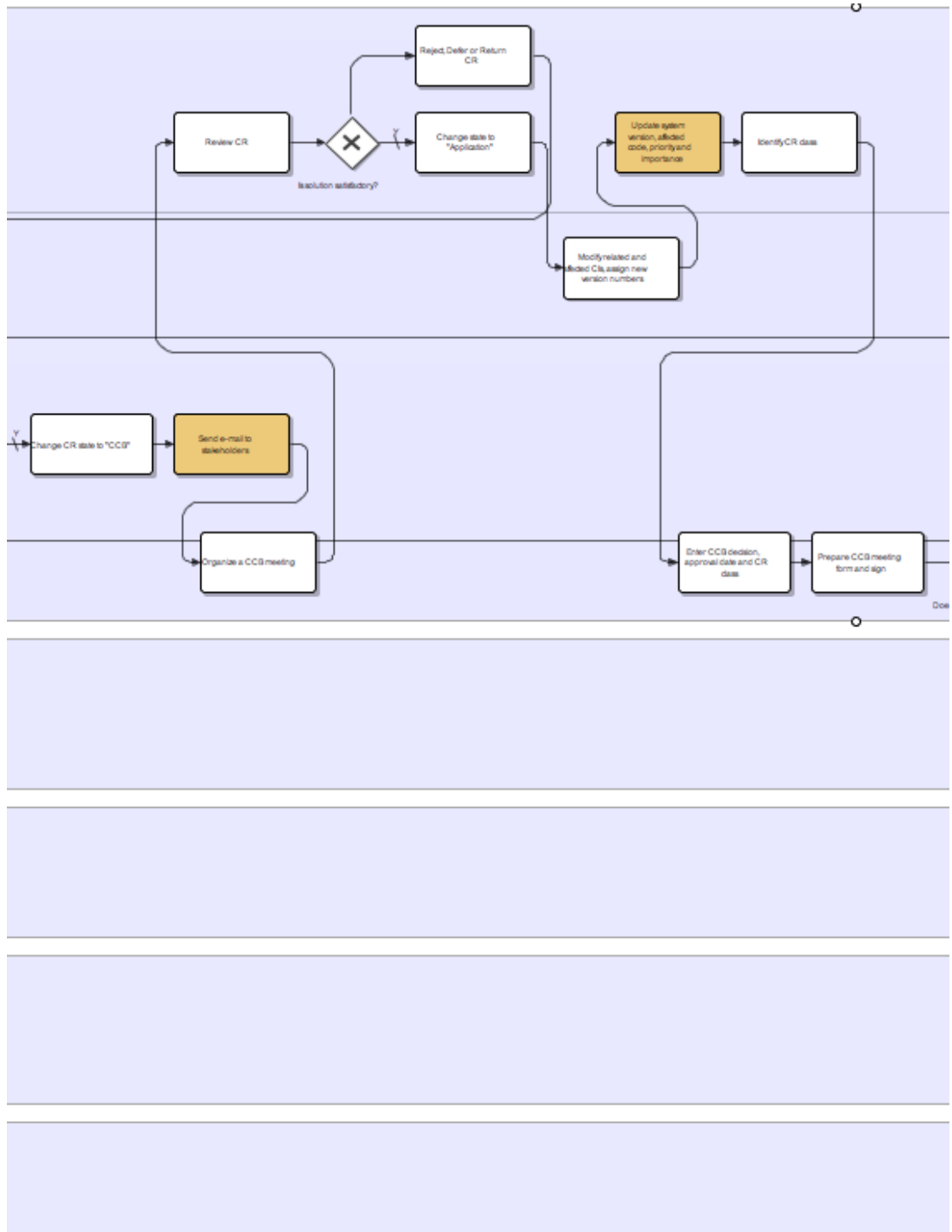


Figure 86 Process Model for KY-020-621-Part 1



**Figure 87 Process Model for KY-020-621-Part 2**



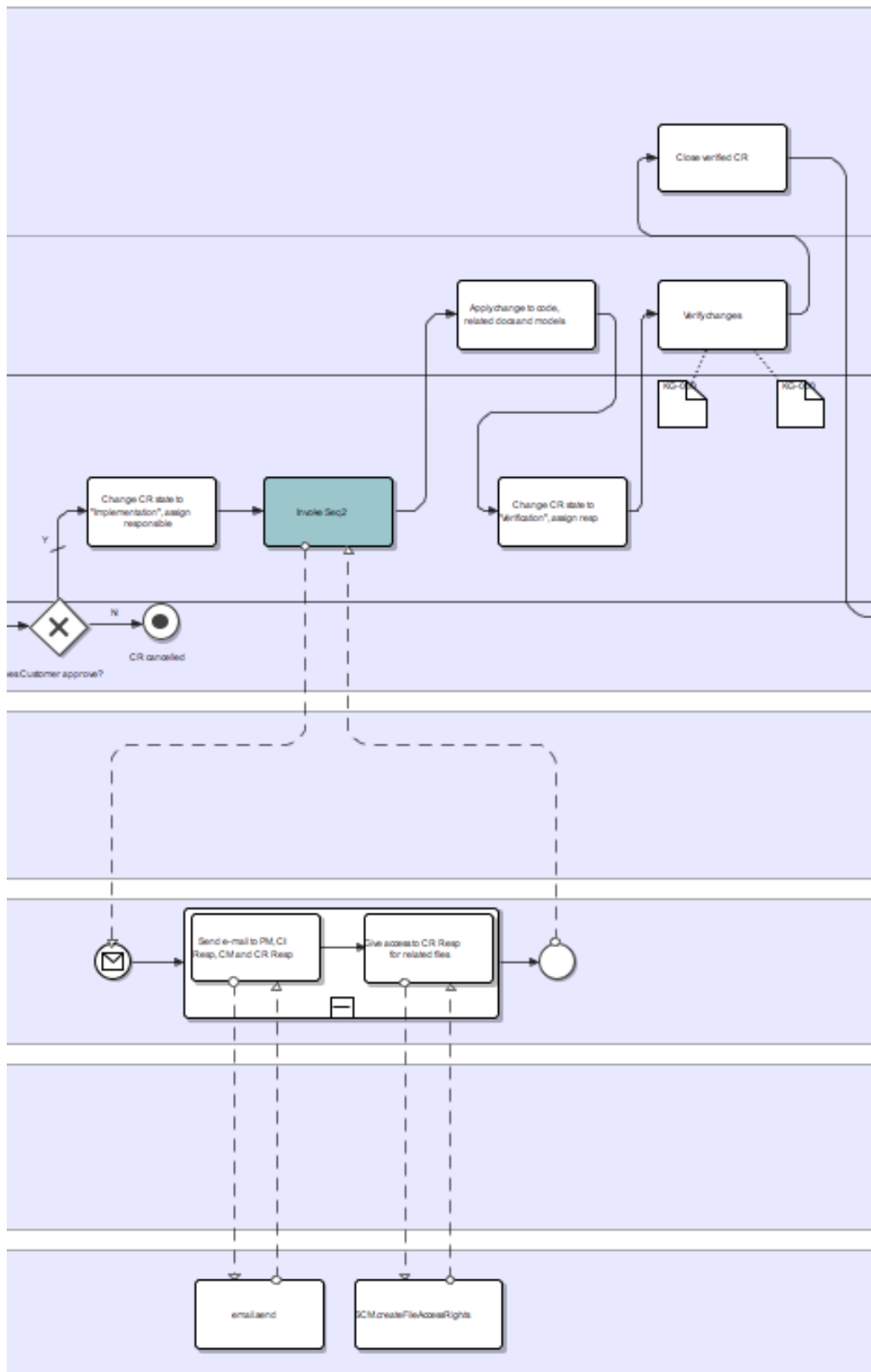


Figure 88 Process Model for KY-020-621-Part 3

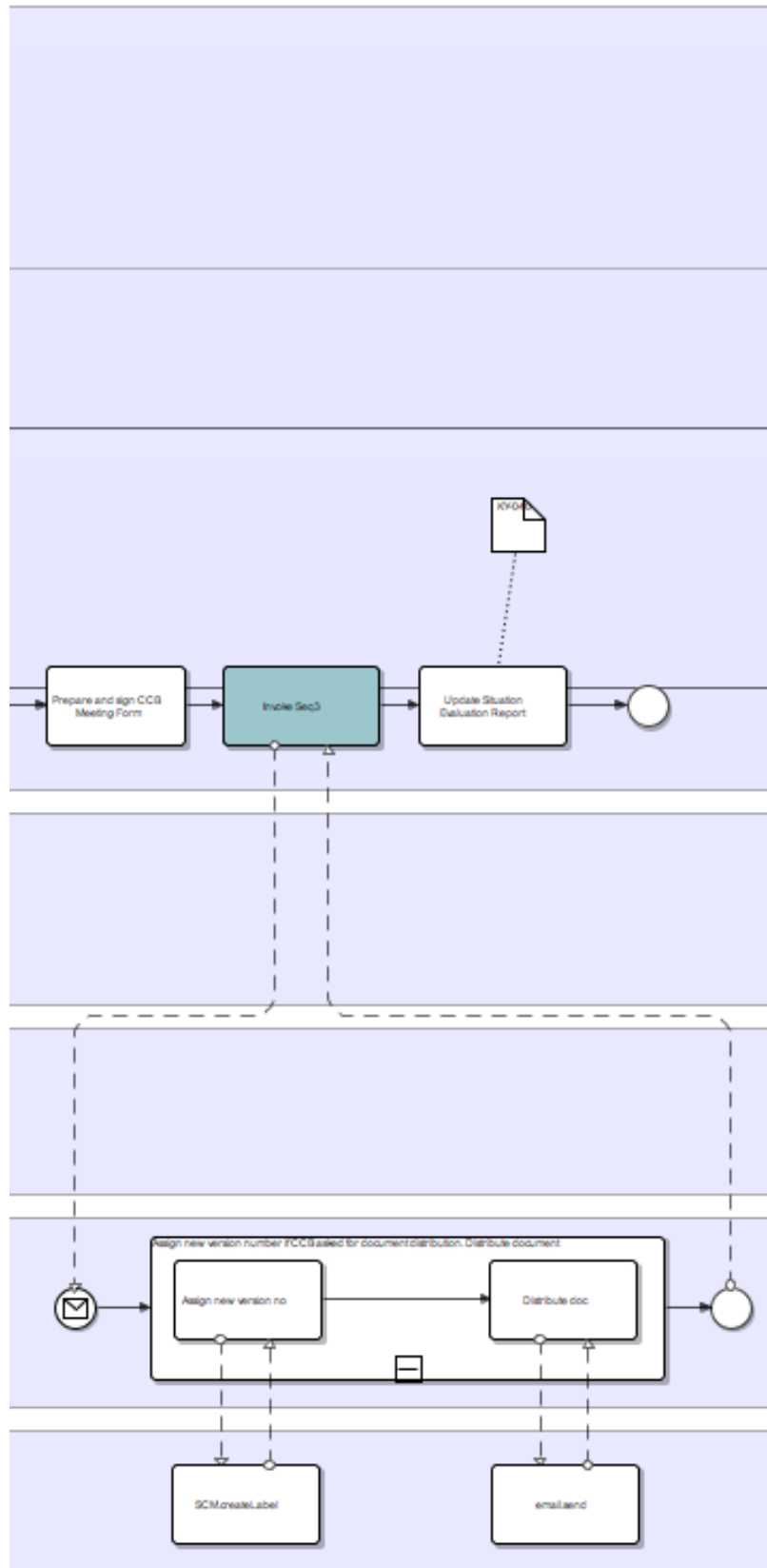


Figure 89 Process Model for KY-020-621-Part 4

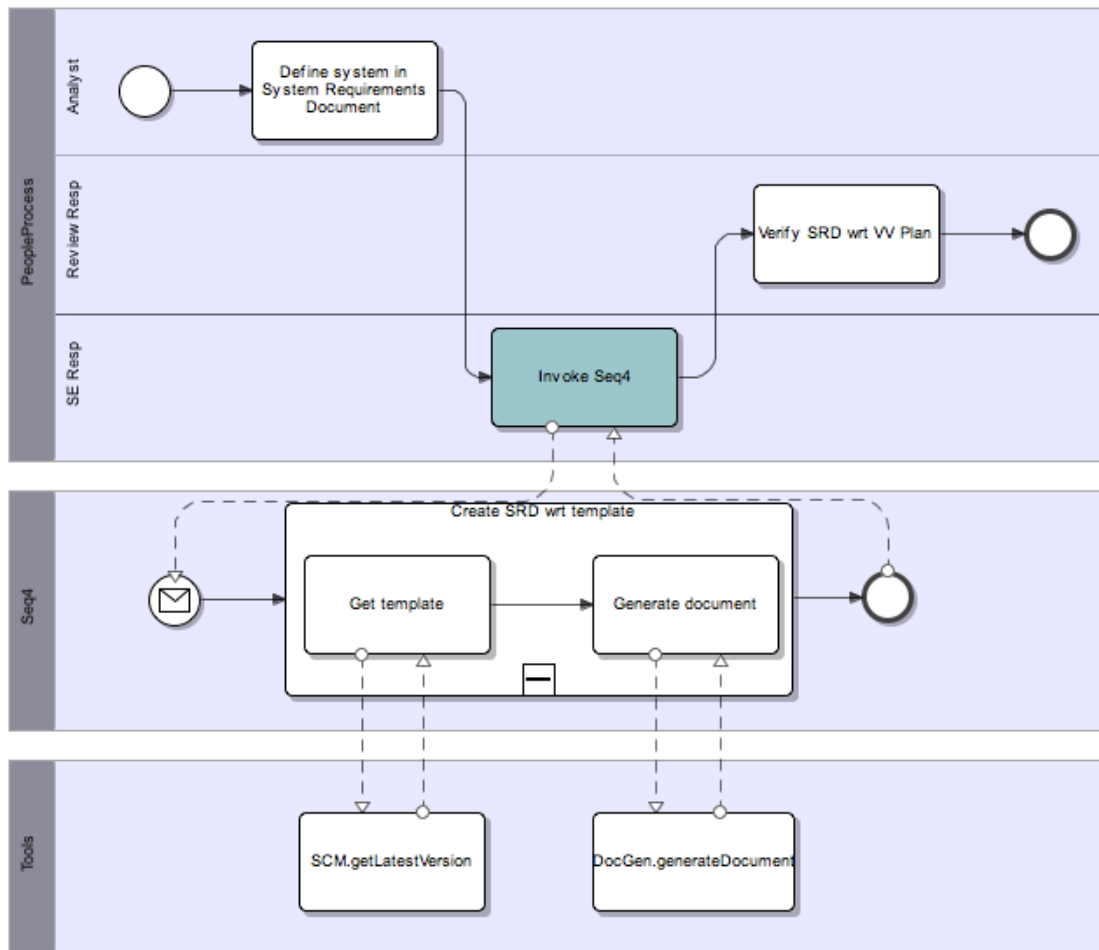
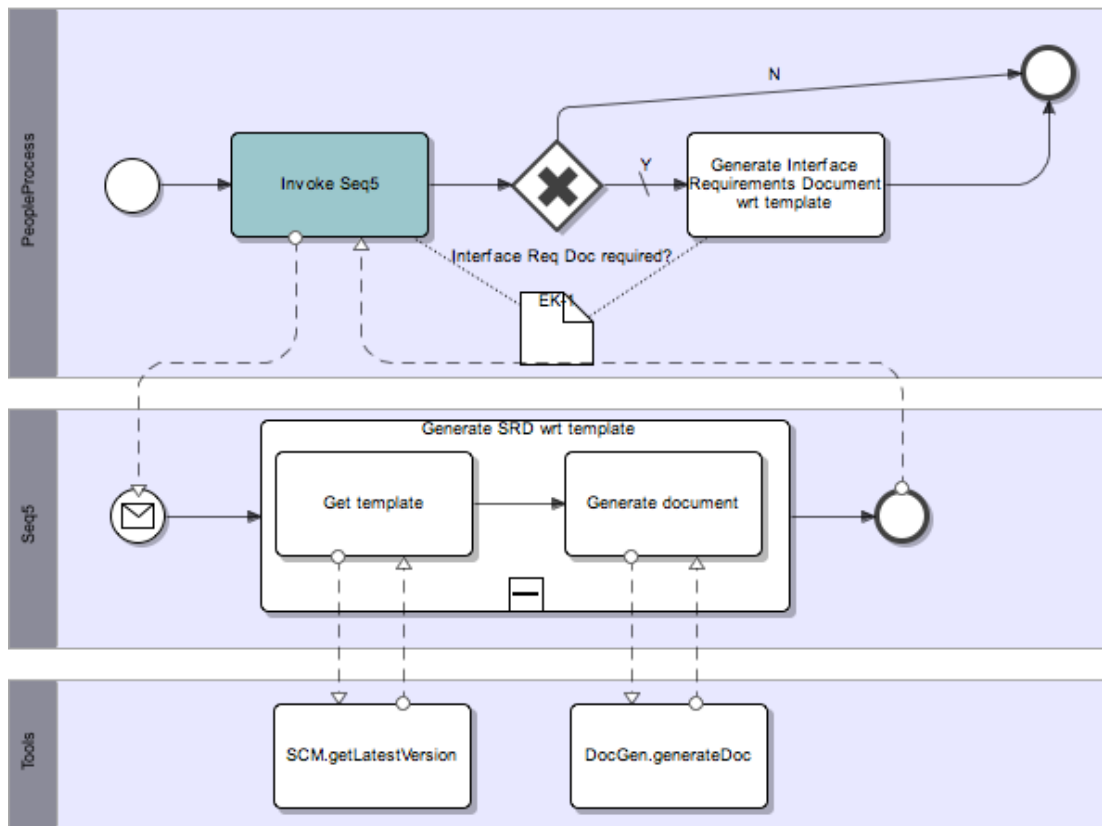
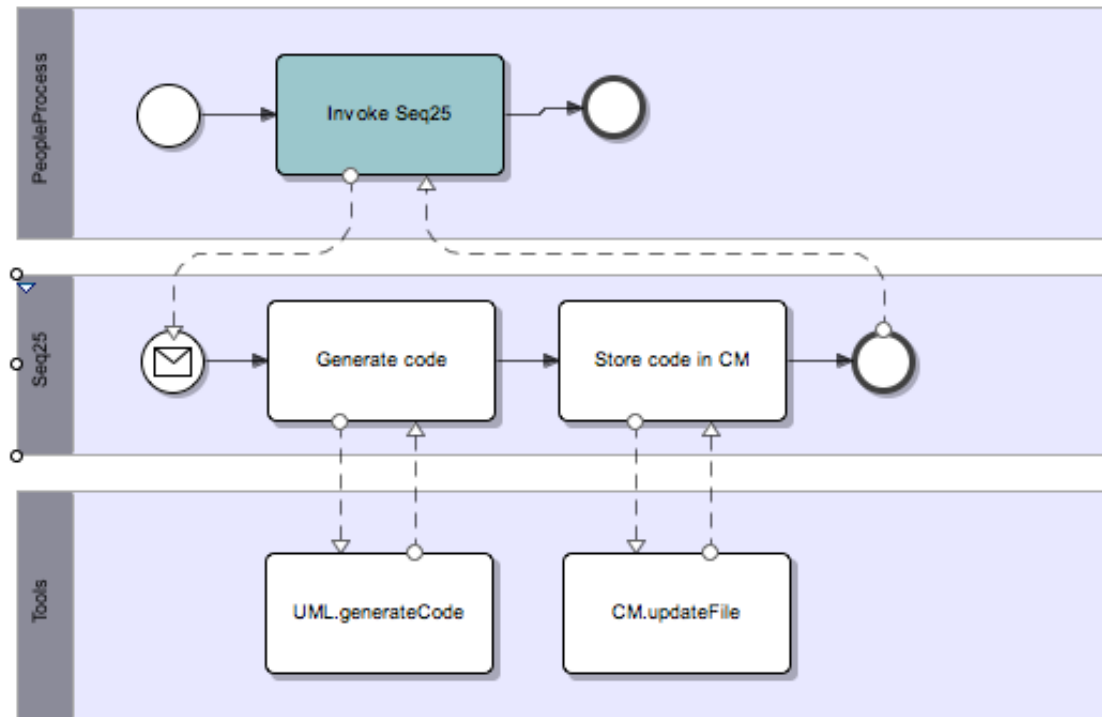


Figure 90 Process Model for UG-010-84



**Figure 91 Process Model for UG-040-83**



**Figure 92 Process Model for UG-070-81**

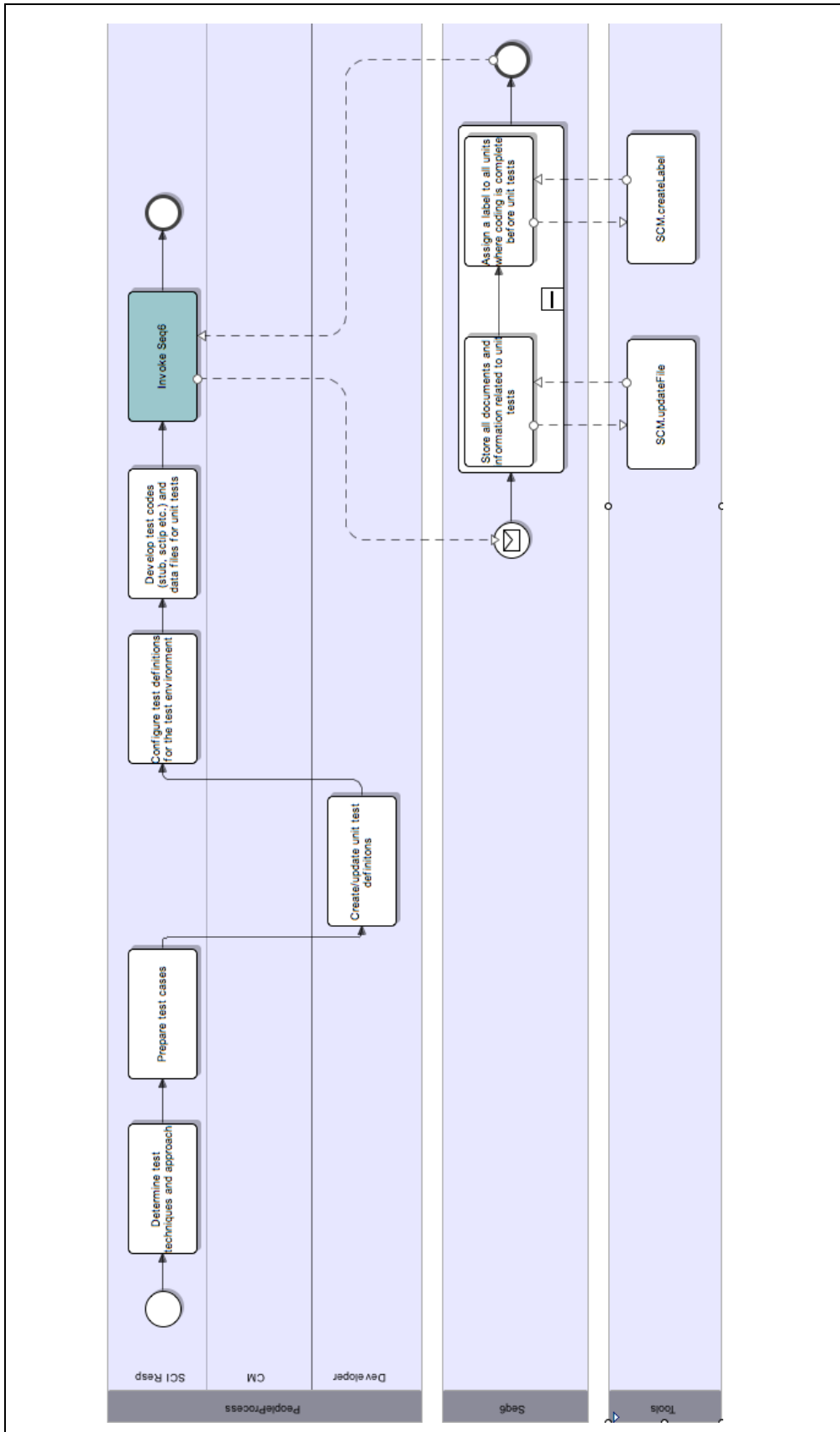


Figure 93 Process Model for UG-070-82

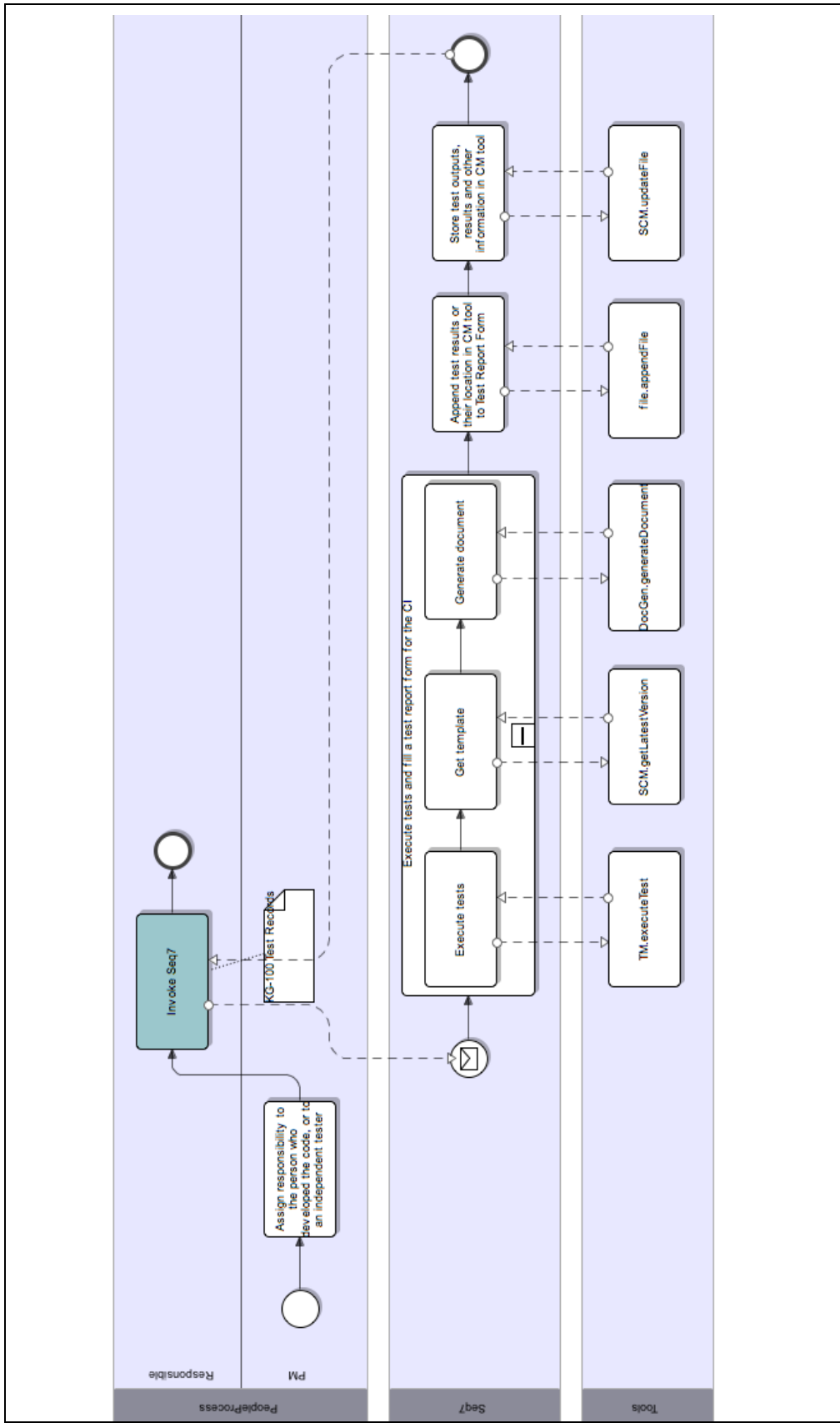


Figure 94 Process Model for UG-070-83

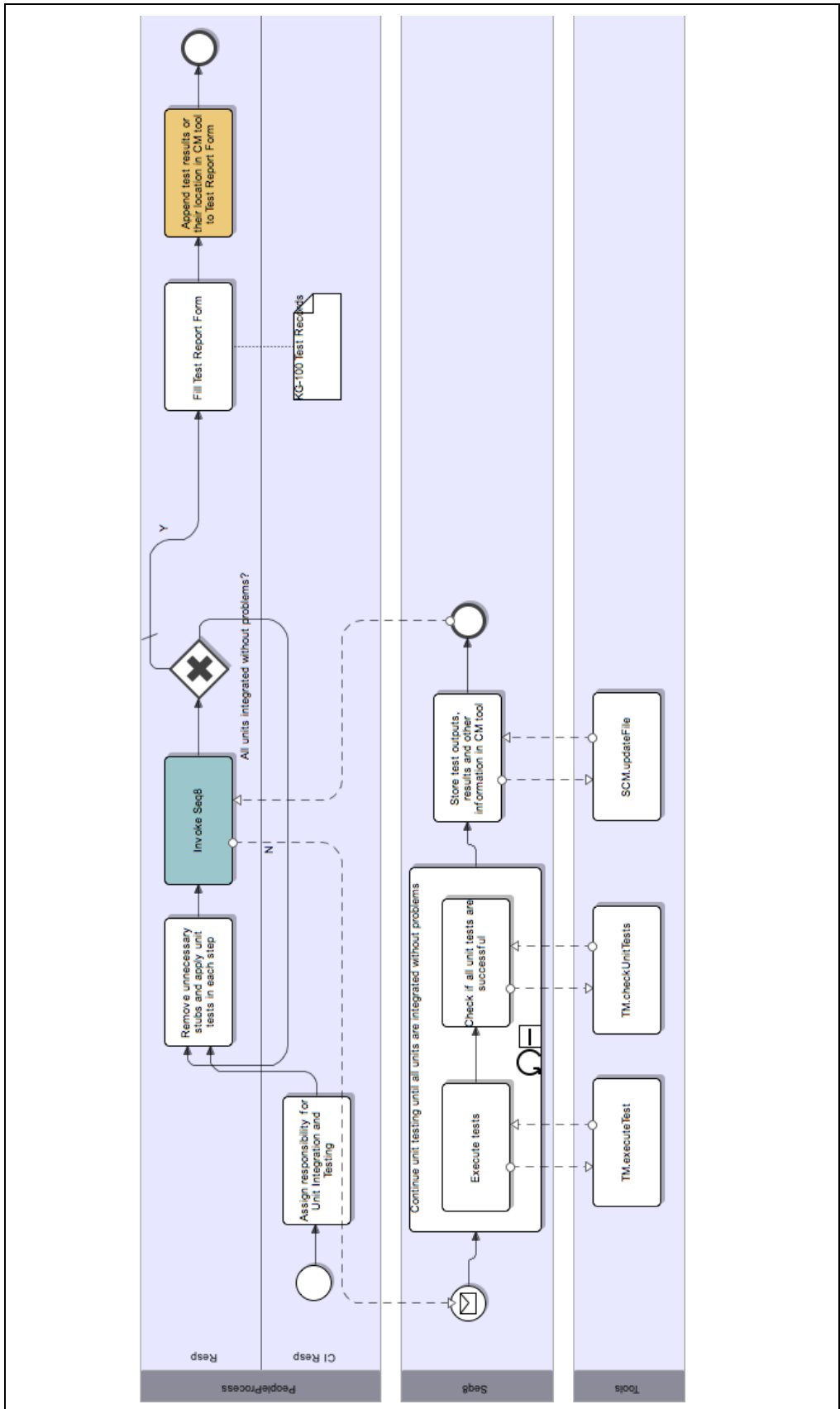


Figure 95 Process Model for UG-070-86



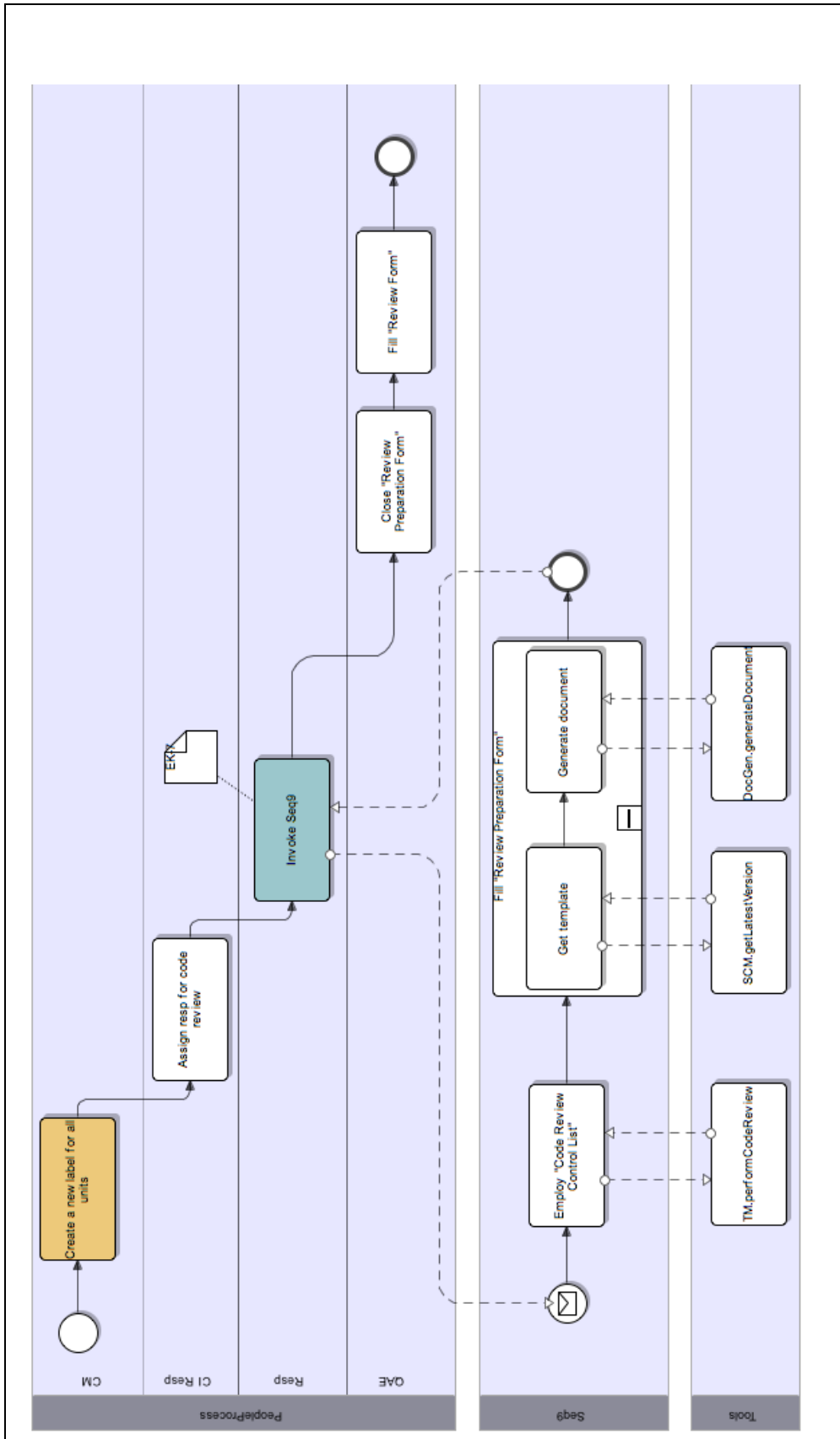
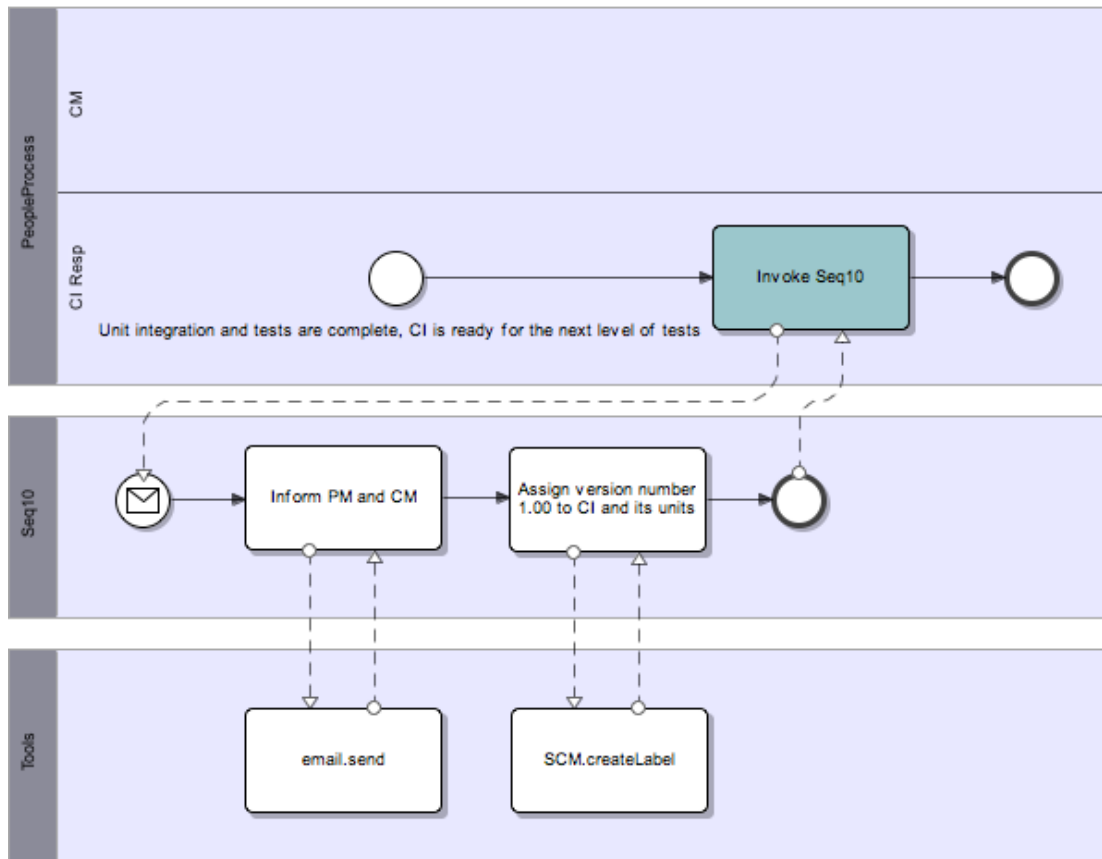


Figure 96 Process Model for UG-070-87



**Figure 97 Procs Model for UG-070-89**

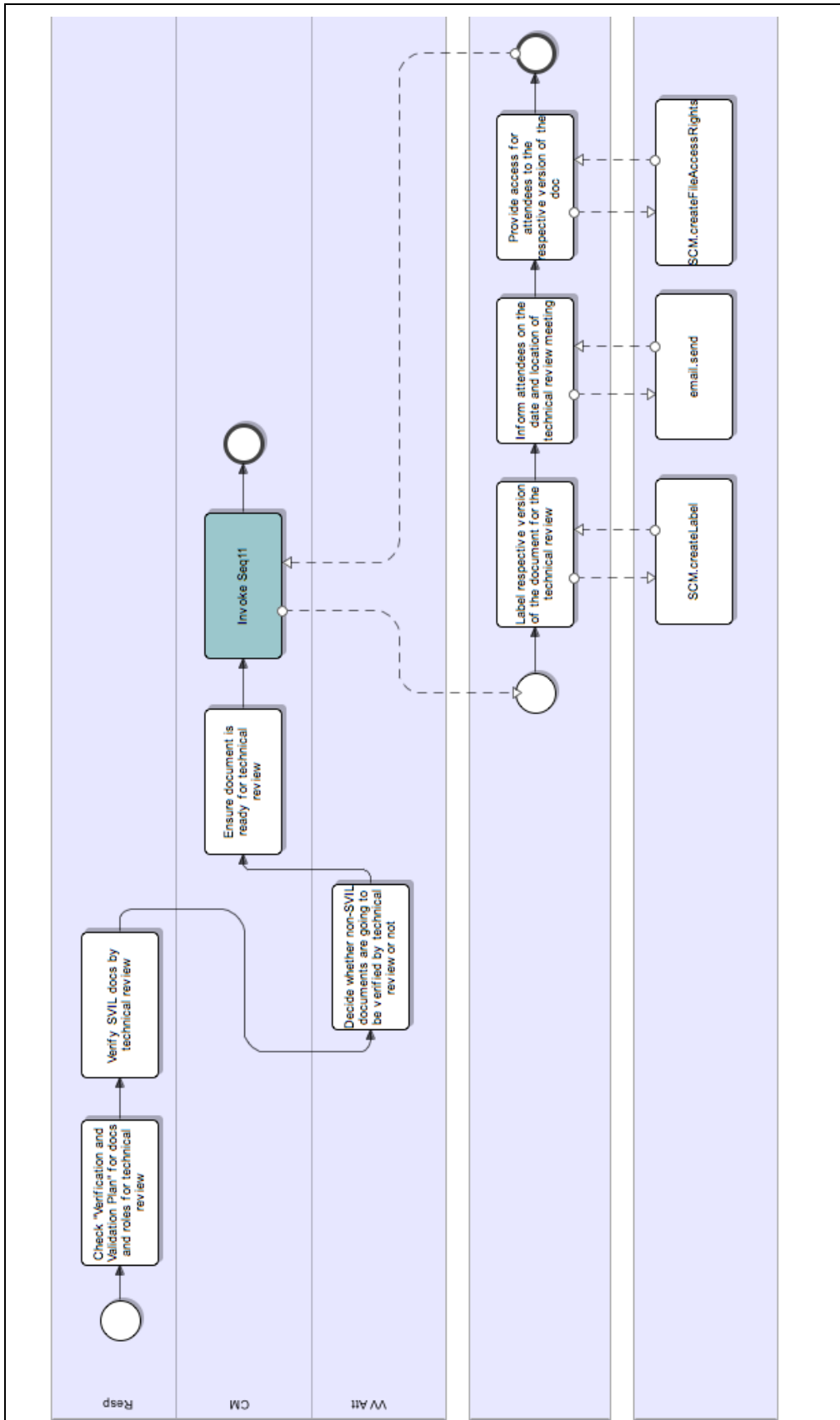


Figure 98 Process Model for UG-190-82

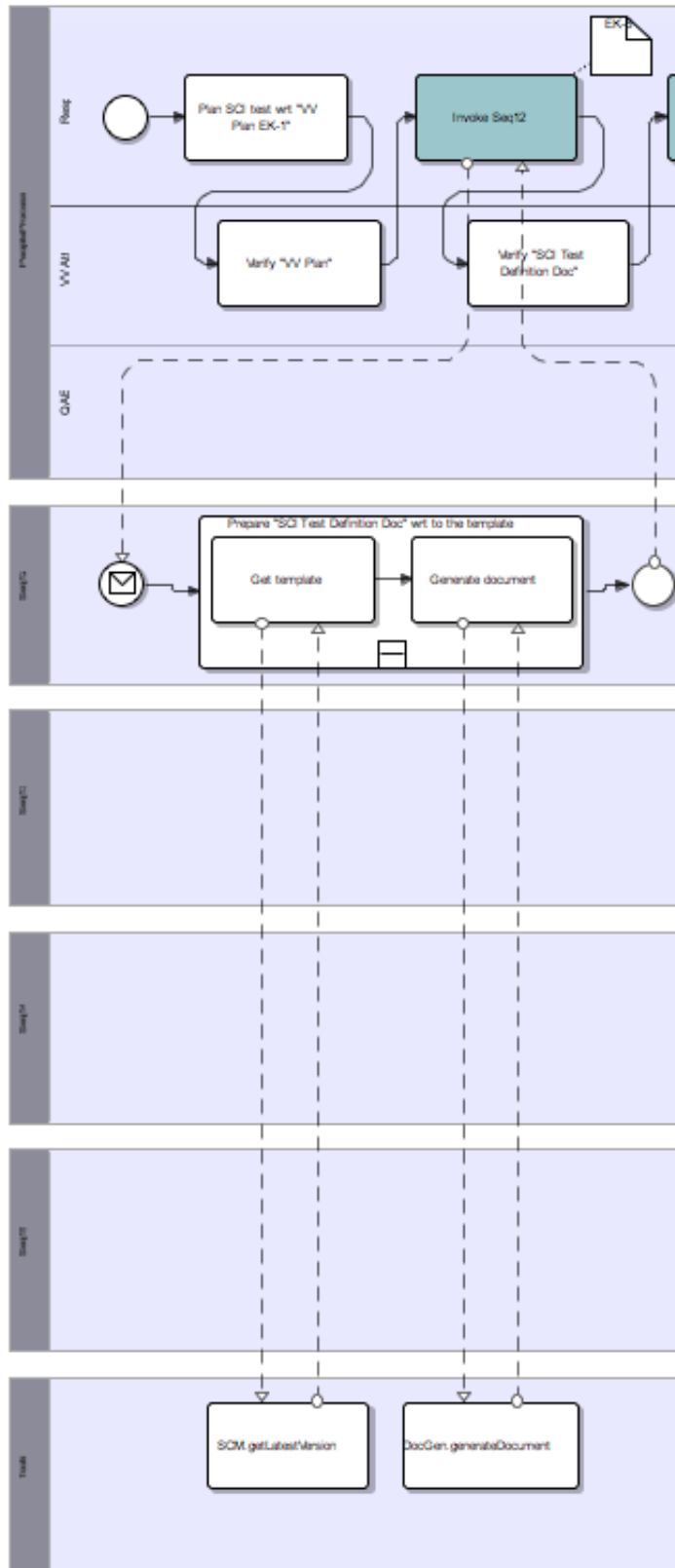
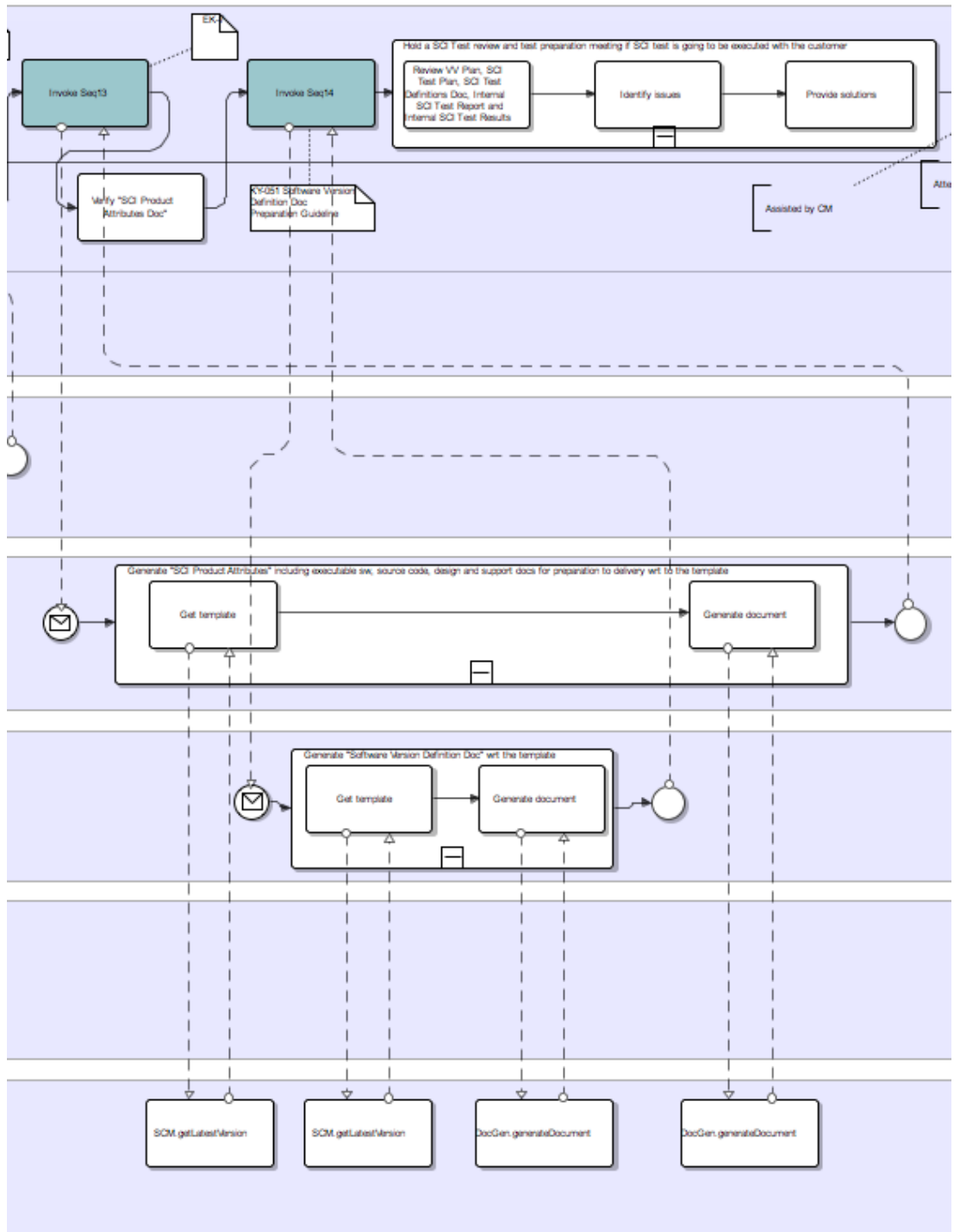
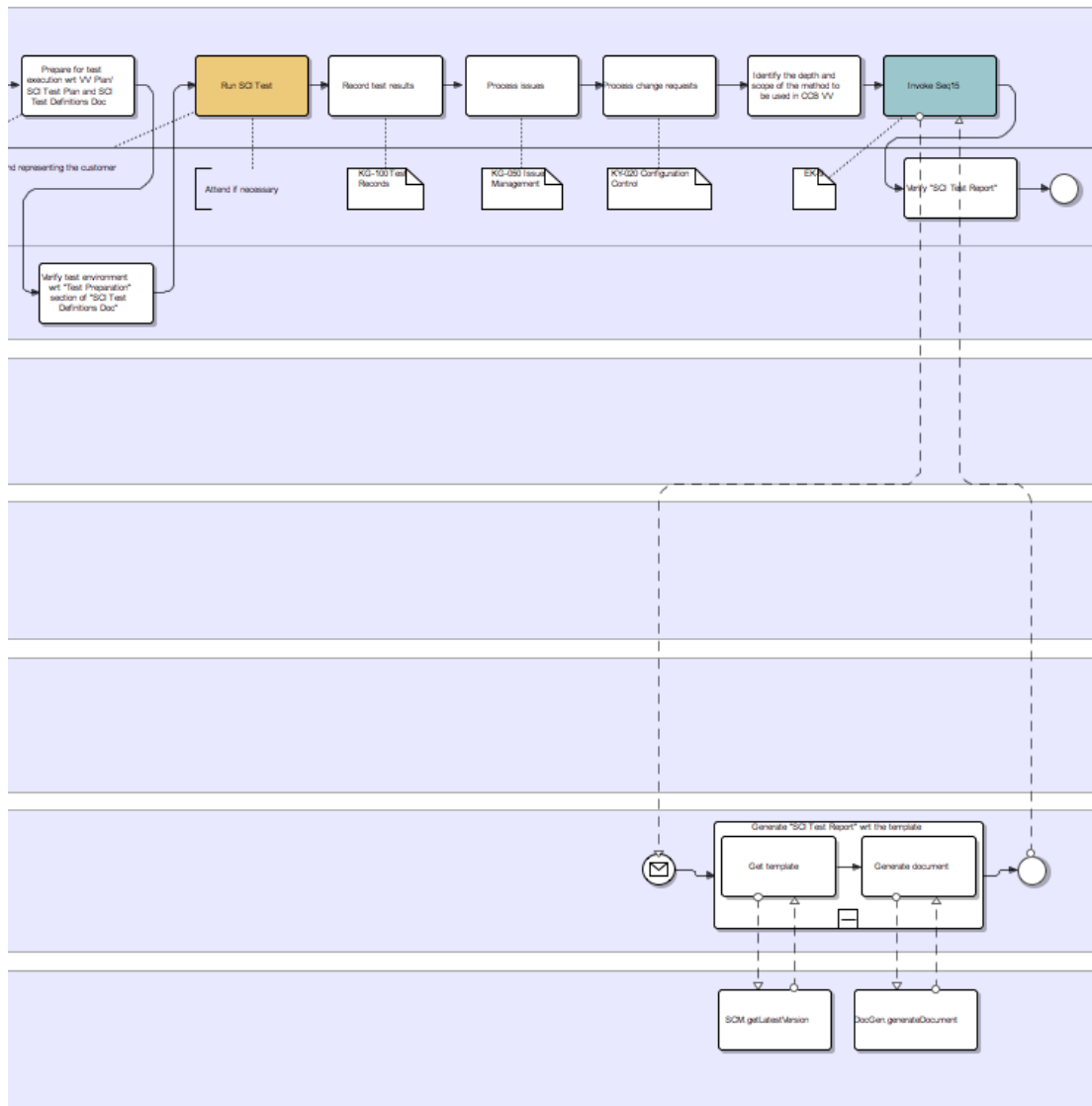


Figure 99 Process Model for UG-190-89-Part1



**Figure 100 Process Model for UG-190-89-Part2**



**Figure 101 Process Model for UG-190-89-Part3**

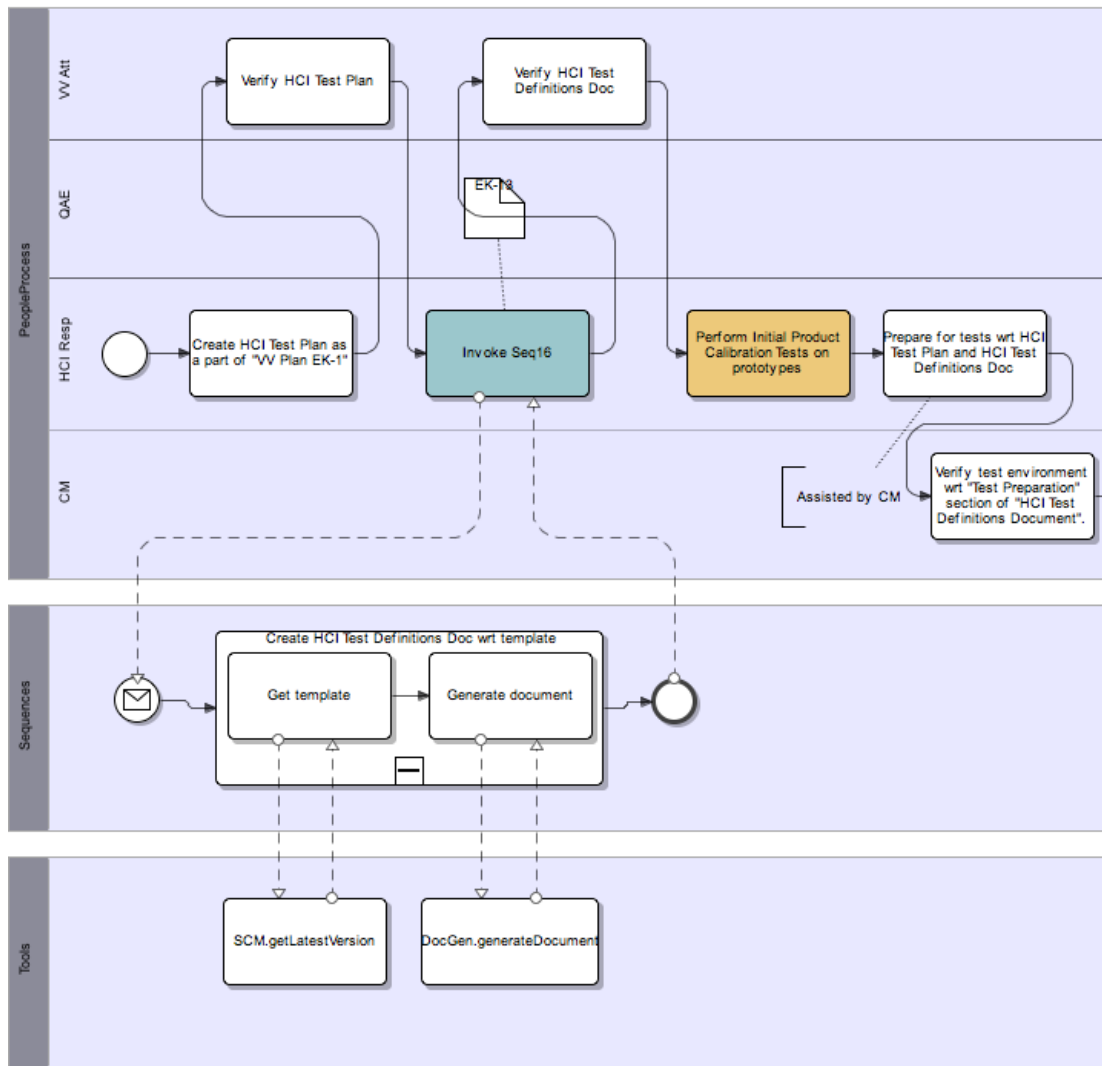


Figure 102 Process Model for UG-190-810-Part 1

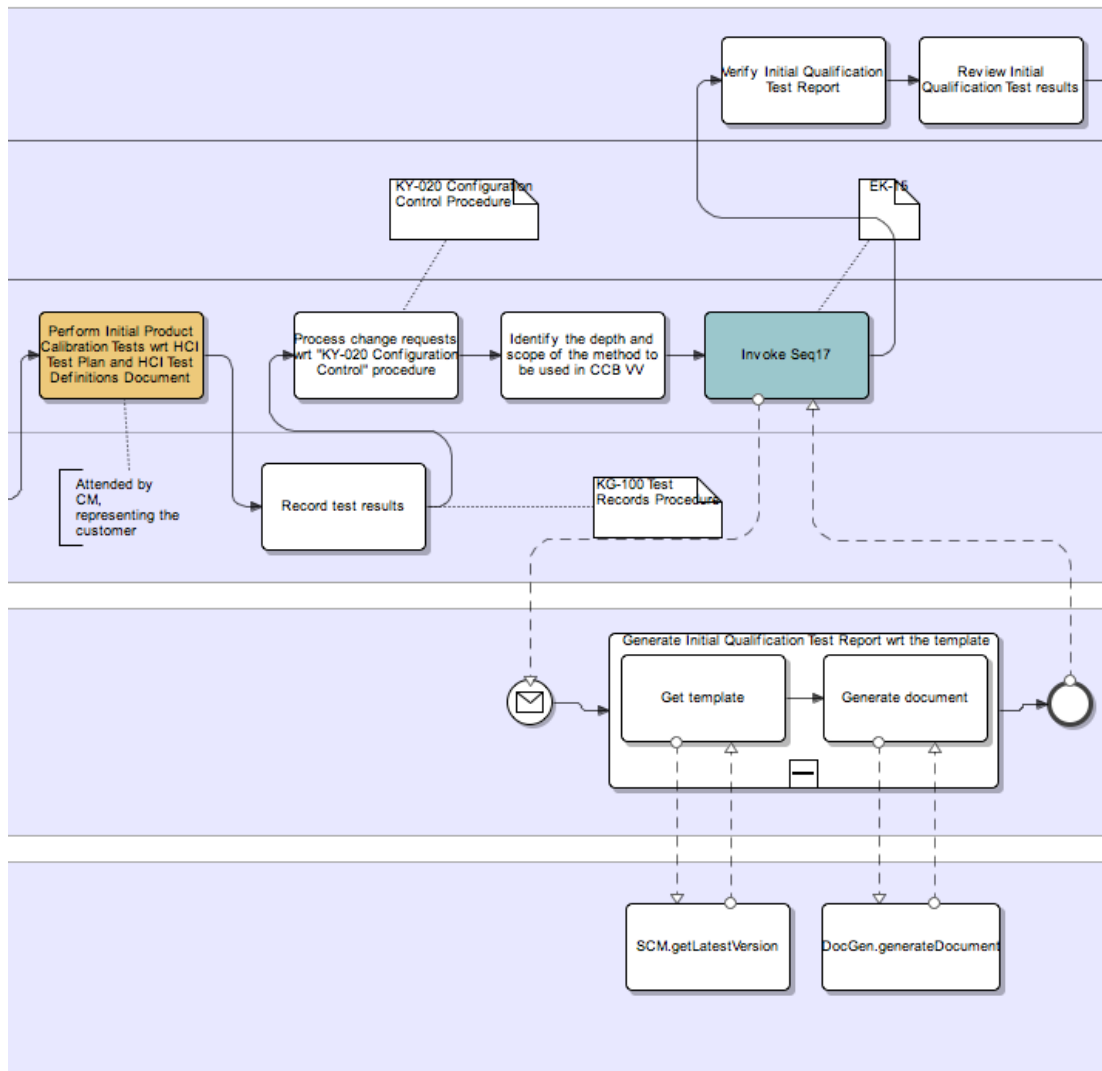
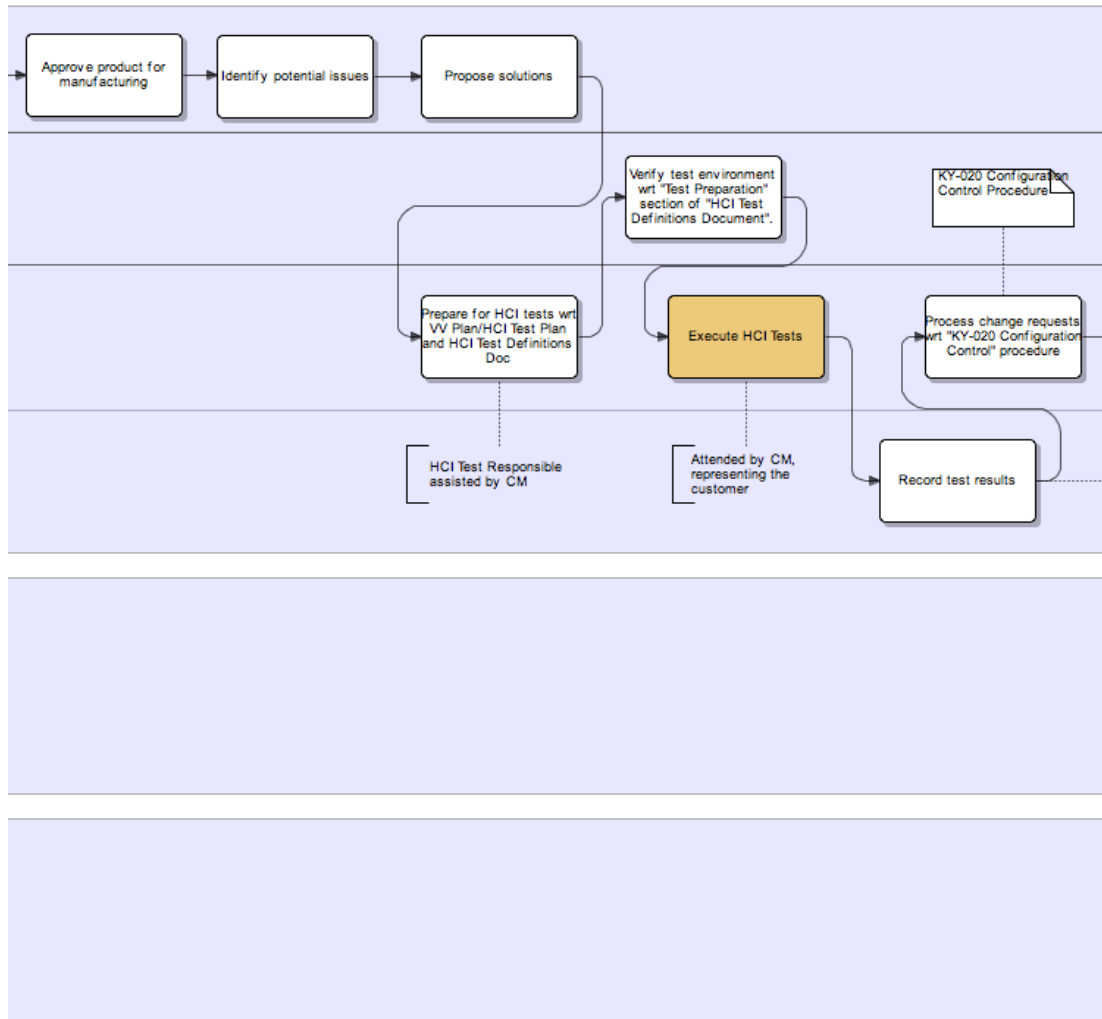


Figure 103 Process Model for UG-190-810-Part 2





**Figure 104 Process Model for UG-190-810-Part 3**

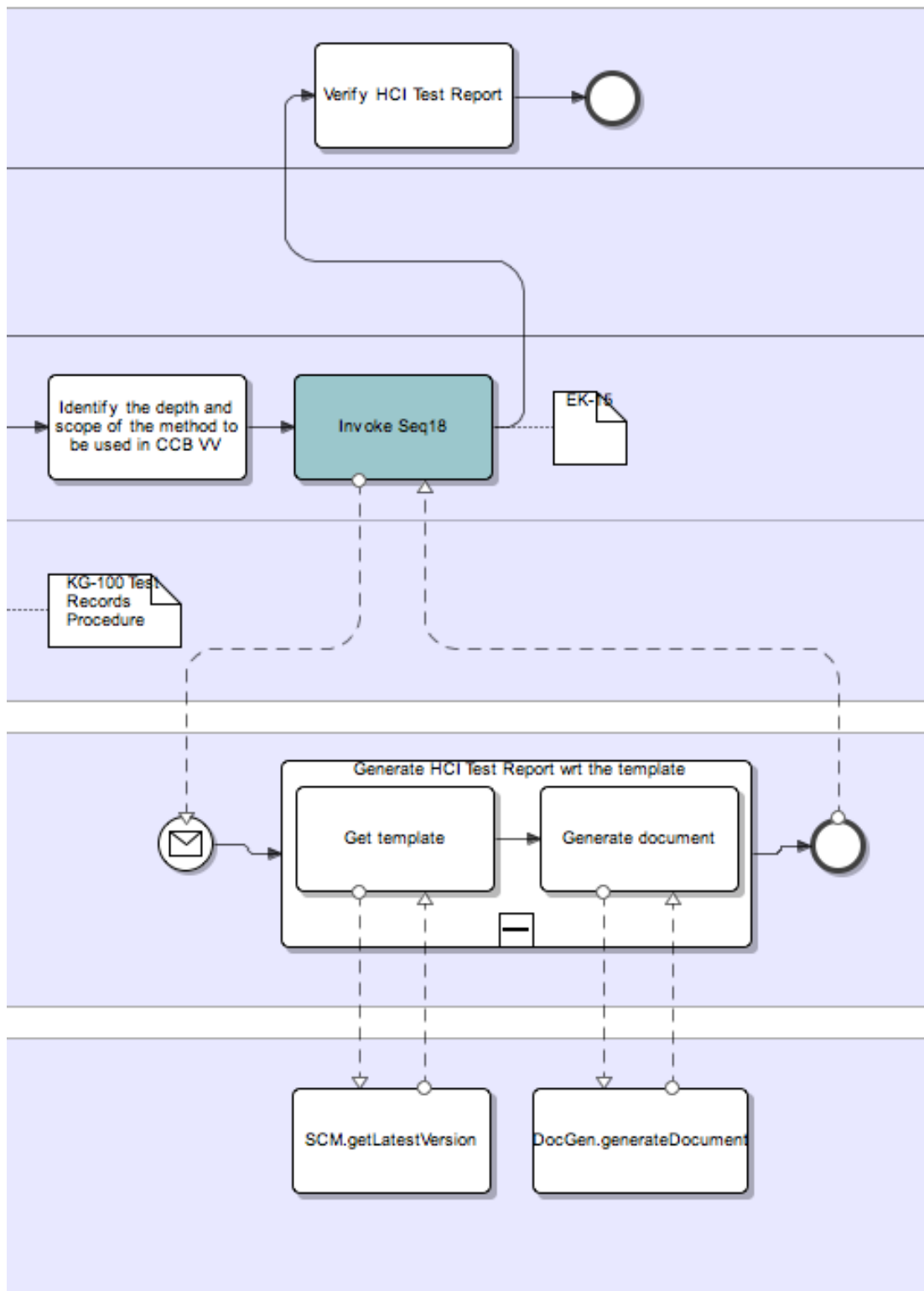


Figure 105 Process Model for UG-190-810-Part 4

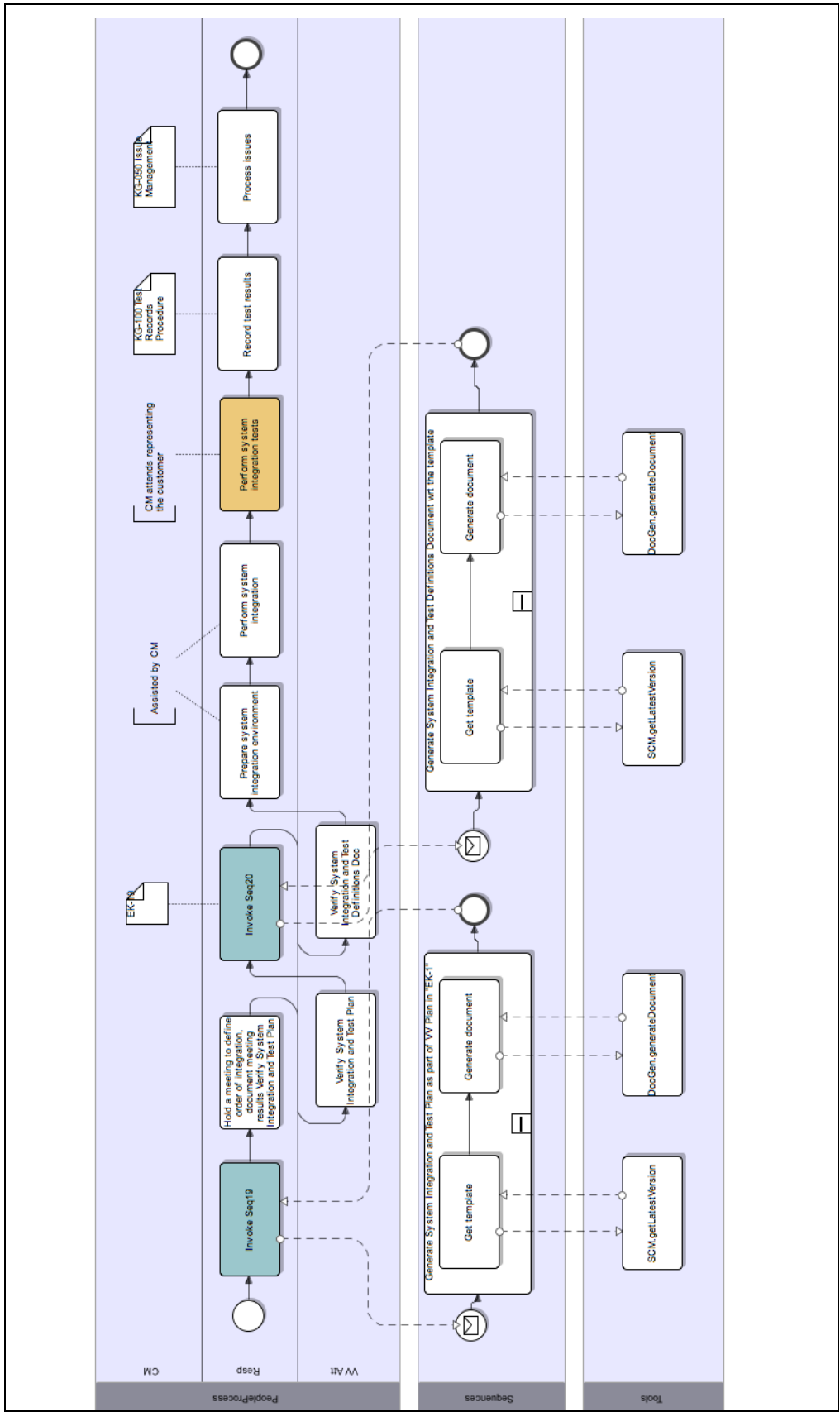


Figure 106 Process Model for UG-190-811

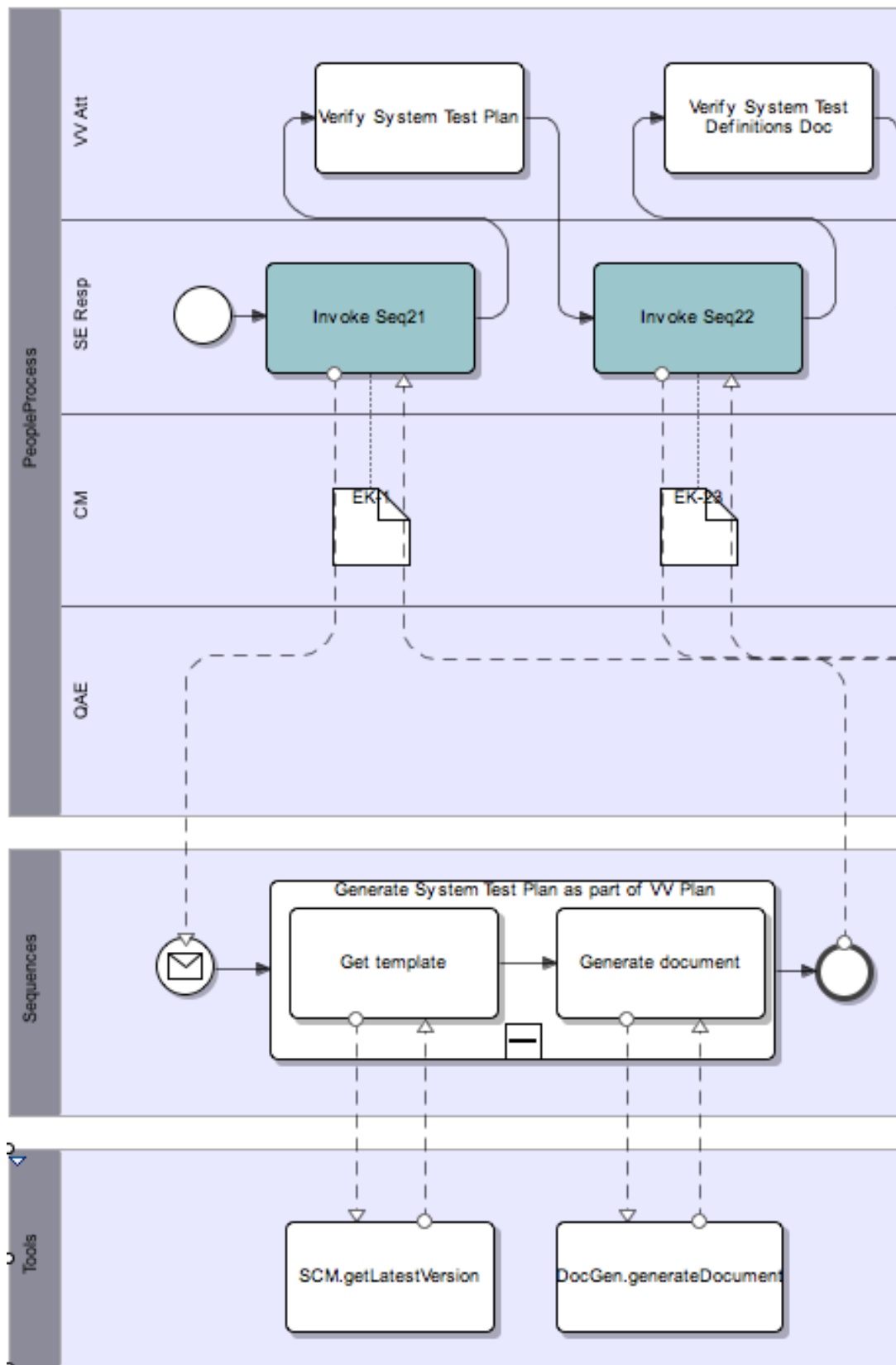


Figure 107 Process Model for UG-190-812-Part 1

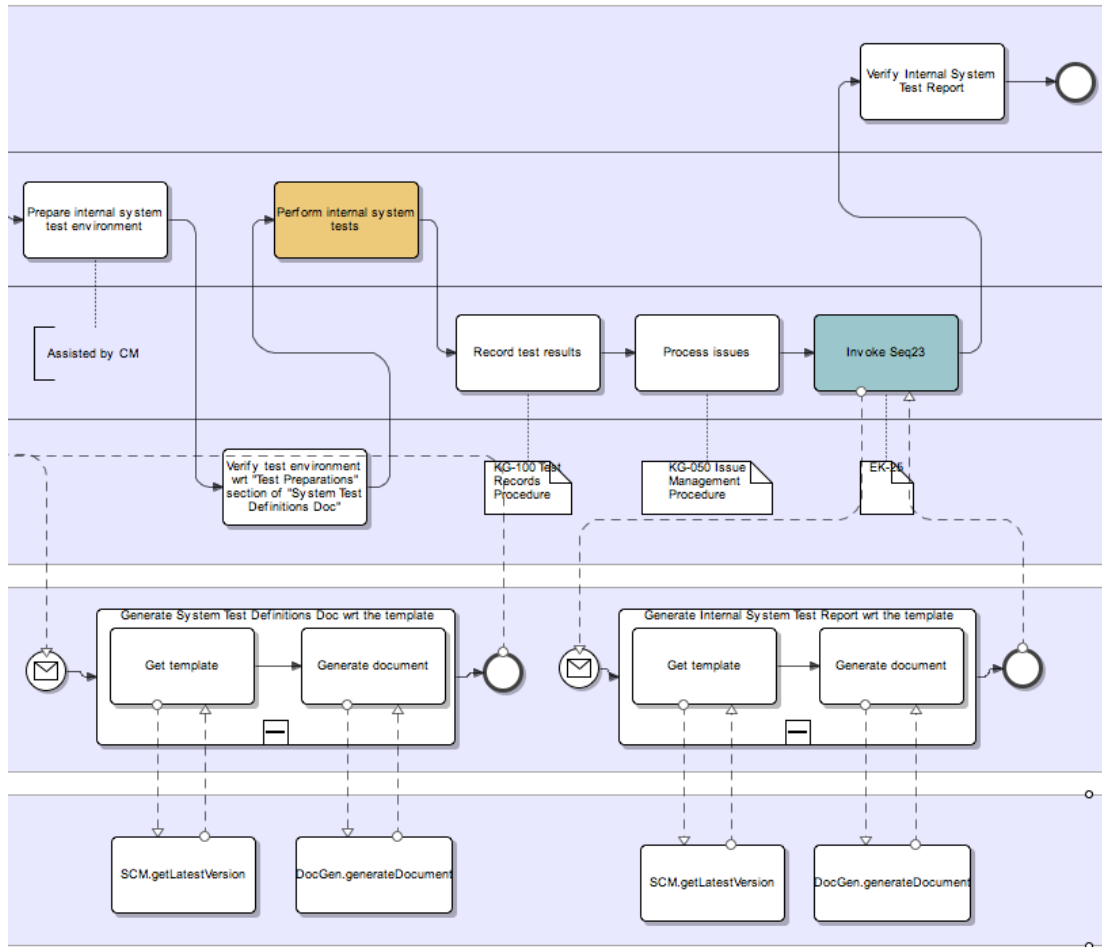


Figure 108 Process Model for UG-190-812-Part 2

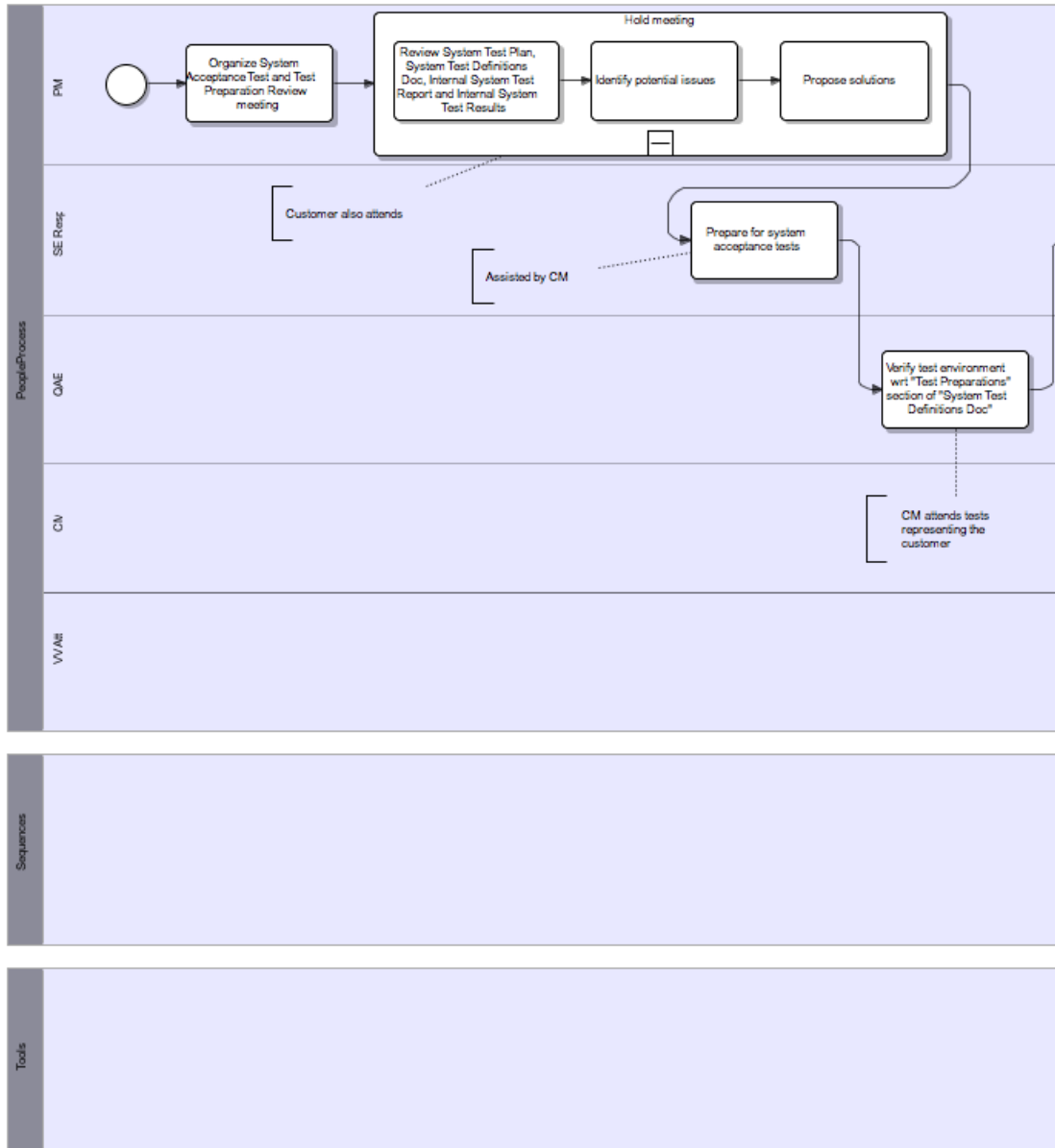
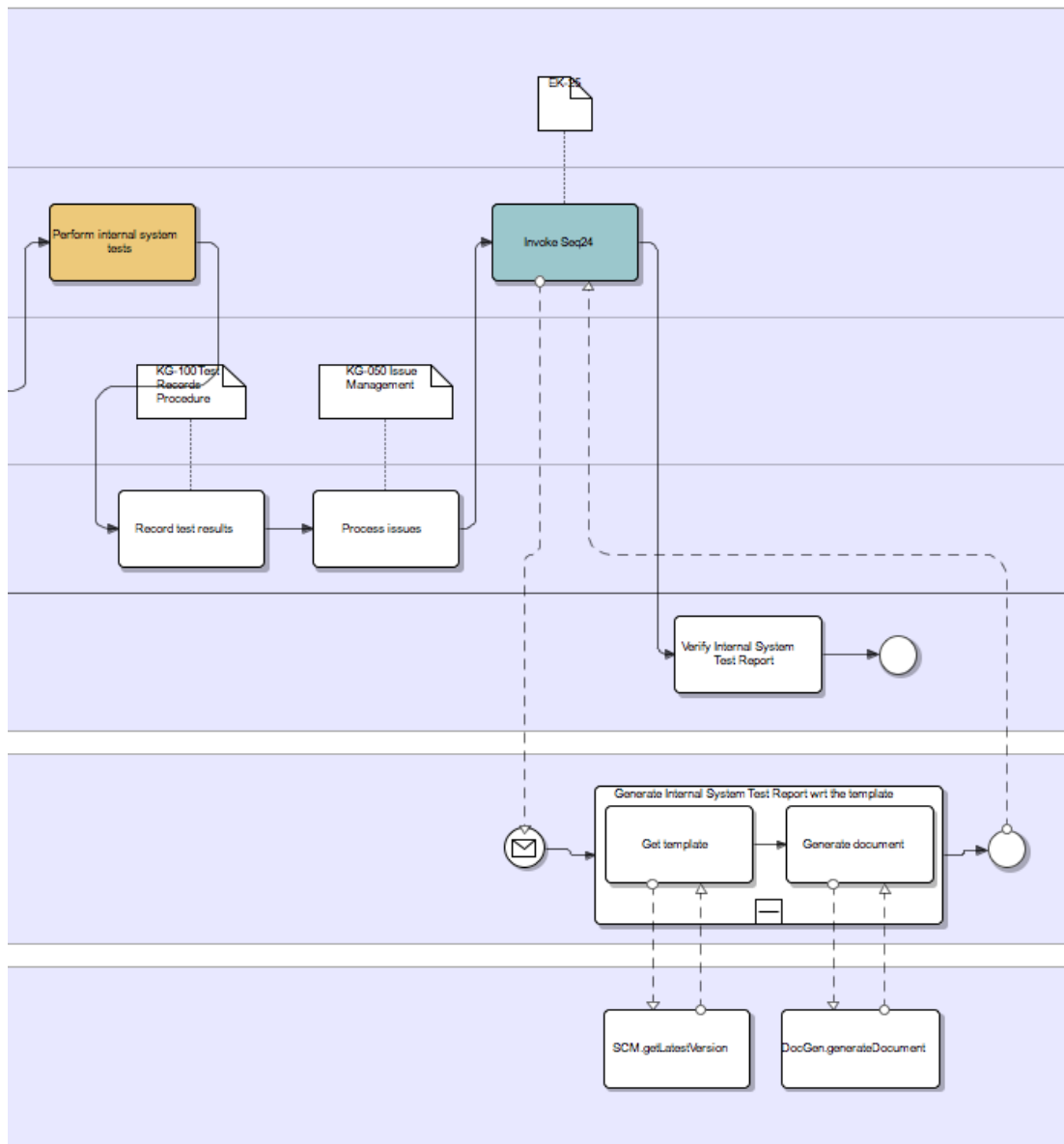


Figure 109 Process Model for UG-190-813-Part 1



**Figure 110 Process Model for UG-190-813-Part 2**

## APPENDIX G: DEFINITIONS FOR WEB SERVICES (CASE STUDY I)

### CaliberRMWrapper WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://caliberrm.ws.tez.alpay.erturkmen.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://caliberrm.ws.tez.alpay.erturkmen.com"
xmlns:intf="http://caliberrm.ws.tez.alpay.erturkmen.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
  <wsdl:types>
    <schema elementFormDefault="qualified"
targetNamespace="http://caliberrm.ws.tez.alpay.erturkmen.com"
xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="createProject">
        <complexType>
          <sequence>
            <element name="serverAddress" type="xsd:string"/>
            <element name="userName" type="xsd:string"/>
            <element name="password" type="xsd:string"/>
            <element name="projectName" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
```



```

<element name="createProjectResponse">
  <complexType>
    <sequence>
      <element name="createProjectReturn" type="xsd:int"/>
    </sequence>
  </complexType>
</element>
<element name="selectProjectByName">
  <complexType>
    <sequence>
      <element name="serverAddress" type="xsd:string"/>
      <element name="userName" type="xsd:string"/>
      <element name="password" type="xsd:string"/>
      <element name="projectName" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="selectProjectByNameResponse">
  <complexType>
    <sequence>
      <element name="selectProjectByNameReturn"
type="xsd:int"/>
    </sequence>
  </complexType>
</element>
<element name="createBaseline">
  <complexType>
    <sequence>
      <element name="serverAddress" type="xsd:string"/>
      <element name="userName" type="xsd:string"/>
      <element name="password" type="xsd:string"/>
      <element name="projectID" type="xsd:int"/>
      <element name="name" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="createBaselineResponse">
  <complexType>
    <sequence>
      <element name="createBaselineReturn" type="xsd:int"/>
    </sequence>
  </complexType>
</element>
</schema>
</wsdl:types>
<wsdl:message name="createProjectRequest">
  <wsdl:part element="impl:createProject"
name="parameters"/>
</wsdl:message>
<wsdl:message name="selectProjectByNameResponse">
  <wsdl:part element="impl:selectProjectByNameResponse"
name="parameters"/>

```

```

</wsdl:message>
<wsdl:message name="selectProjectByNameRequest">
  <wsdl:part element="impl:selectProjectByName"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createProjectResponse">
  <wsdl:part element="impl:createProjectResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createBaselineResponse">
  <wsdl:part element="impl:createBaselineResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createBaselineRequest">
  <wsdl:part element="impl:createBaseline"
name="parameters"/>
</wsdl:message>
<wsdl:portType name="CaliberRMWrapper">

  <wsdl:operation name="createProject">
    <wsdl:input message="impl:createProjectRequest"
name="createProjectRequest"/>
    <wsdl:output message="impl:createProjectResponse"
name="createProjectResponse"/>
  </wsdl:operation>
  <wsdl:operation name="selectProjectByName">
    <wsdl:input message="impl:selectProjectByNameRequest"
name="selectProjectByNameRequest"/>
    <wsdl:output
message="impl:selectProjectByNameResponse"
name="selectProjectByNameResponse"/>
  </wsdl:operation>
  <wsdl:operation name="createBaseline">
    <wsdl:input message="impl:createBaselineRequest"
name="createBaselineRequest"/>
    <wsdl:output message="impl:createBaselineResponse"
name="createBaselineResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="CaliberRMWrapperSoapBinding"
type="impl:CaliberRMWrapper">
  <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="createProject">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="createProjectRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="createProjectResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>

```

```

    <wsdl:operation name="selectProjectByName">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="selectProjectByNameRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="selectProjectByNameResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="createBaseline">
      <wsdlsoap:operation soapAction=""/>
      <wsdl:input name="createBaselineRequest">
        <wsdlsoap:body use="literal"/>
      </wsdl:input>
      <wsdl:output name="createBaselineResponse">
        <wsdlsoap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="CaliberRMWrapperService">
    <wsdl:port binding="impl:CaliberRMWrapperSoapBinding"
name="CaliberRMWrapper">
      <wsdlsoap:address
location="http://localhost:8081/BorlandCaliberWS/services/CaliberRMWrapper"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>

```

## StarTeamWrapper WSDL

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://starteam.ws.tez.alpay.erturkmen.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://starteam.ws.tez.alpay.erturkmen.com"
xmlns:intf="http://starteam.ws.tez.alpay.erturkmen.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
  <wsdl:types>
    <schema elementFormDefault="qualified"
targetNamespace="http://starteam.ws.tez.alpay.erturkmen.com"
xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="login">
        <complexType>
          <sequence>
            <element name="url" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </wsdl:types>

```

```

    </complexType>
  </element>
  <element name="loginResponse">
    <complexType>
      <sequence>
        <element name="loginReturn" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="getProjects">
    <complexType>
      <sequence>
        <element name="url" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="getProjectsResponse">
    <complexType>
      <sequence>
        <element maxOccurs="unbounded" name="getProjectsReturn"
type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="createLink">
    <complexType>
      <sequence>
        <element name="url1" type="xsd:string"/>
        <element name="item1ID" type="xsd:int"/>
        <element name="item1Type" type="xsd:string"/>
        <element name="url2" type="xsd:string"/>
        <element name="item2ID" type="xsd:int"/>
        <element name="item2Type" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="createLinkResponse">
    <complexType>
      <sequence>
        <element name="createLinkReturn" type="xsd:int"/>
      </sequence>
    </complexType>
  </element>
  <element name="getLatestVersion">
    <complexType>
      <sequence>
        <element name="url" type="xsd:string"/>
        <element name="filename" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="getLatestVersionResponse">

```

```

    <complexType>
      <sequence>
        <element name="getLatestVersionReturn"
type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="createTask">
    <complexType>
      <sequence>
        <element name="url" type="xsd:string"/>
        <element name="name" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="createTaskResponse">
    <complexType>
      <sequence>
        <element name="createTaskReturn" type="xsd:int"/>
      </sequence>
    </complexType>
  </element>
  <element name="createLabel">
    <complexType>
      <sequence>
        <element name="url" type="xsd:string"/>
        <element name="labelBaseName" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="createLabelResponse">
    <complexType>
      <sequence>
        <element name="createLabelReturn" type="xsd:int"/>
      </sequence>
    </complexType>
  </element>
  <element name="createProject">
    <complexType>
      <sequence>
        <element name="serverAddress" type="xsd:string"/>
        <element name="username" type="xsd:string"/>
        <element name="password" type="xsd:string"/>
        <element name="projectName" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="createProjectResponse">
    <complexType>
      <sequence>
        <element name="createProjectReturn" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>

```

```

    </complexType>
  </element>
  <element name="selectProjectByName">
    <complexType>
      <sequence>
        <element name="url" type="xsd:string"/>
        <element name="projectName" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="selectProjectByNameResponse">
    <complexType>
      <sequence>
        <element name="selectProjectByNameReturn"
type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="updateFile">
    <complexType>
      <sequence>
        <element name="url" type="xsd:string"/>
        <element name="filename" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="updateFileResponse">
    <complexType>
      <sequence>
        <element name="updateFileReturn" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="createLabelForFile">
    <complexType>
      <sequence>
        <element name="url" type="xsd:string"/>
        <element name="filename" type="xsd:string"/>
      </sequence>
    </complexType>
  </element>
  <element name="createLabelForFileResponse">
    <complexType>
      <sequence>
        <element name="createLabelForFileReturn"
type="xsd:int"/>
      </sequence>
    </complexType>
  </element>
  <element name="freezeLabel">
    <complexType>
      <sequence>

```

```

        <element name="url" type="xsd:string"/>
        <element name="labelID" type="xsd:int"/>
    </sequence>
</complexType>
</element>
<element name="freezeLabelResponse">
    <complexType>
        <sequence>
            <element name="freezeLabelReturn" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="getLinkedItems">
    <complexType>
        <sequence>
            <element name="url" type="xsd:string"/>
            <element name="itemID" type="xsd:int"/>
            <element name="itemType" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
<element name="getLinkedItemsResponse">
    <complexType>
        <sequence>
            <element maxOccurs="unbounded"
name="getLinkedItemsReturn" type="xsd:int"/>
        </sequence>
    </complexType>
</element>
<element name="setCRStatus">
    <complexType>
        <sequence>
            <element name="url" type="xsd:string"/>
            <element maxOccurs="unbounded" name="itemIDs"
type="xsd:int"/>
            <element name="status" type="xsd:int"/>
        </sequence>
    </complexType>
</element>
<element name="setCRStatusResponse">
    <complexType/>
</element>
</schema>
</wsdl:types>
<wsdl:message name="setCRStatusResponse">
    <wsdl:part element="impl:setCRStatusResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createLinkRequest">
    <wsdl:part element="impl:createLink" name="parameters"/>
</wsdl:message>
<wsdl:message name="getProjectsRequest">

```

```

        <wsdl:part element="impl:getProjects"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="freezeLabelResponse">
        <wsdl:part element="impl:freezeLabelResponse"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="loginRequest">
        <wsdl:part element="impl:login" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="getLinkedItemsRequest">
        <wsdl:part element="impl:getLinkedItems"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="updateFileRequest">
        <wsdl:part element="impl:updateFile" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="updateFileResponse">
        <wsdl:part element="impl:updateFileResponse"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="getLinkedItemsResponse">
        <wsdl:part element="impl:getLinkedItemsResponse"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="createTaskResponse">
        <wsdl:part element="impl:createTaskResponse"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="createLabelRequest">
        <wsdl:part element="impl:createLabel"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="getProjectsResponse">
        <wsdl:part element="impl:getProjectsResponse"
name="parameters"/>
    </wsdl:message>

    <wsdl:message name="createTaskRequest">
        <wsdl:part element="impl:createTask" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="getLatestVersionRequest">
        <wsdl:part element="impl:getLatestVersion"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="selectProjectByNameResponse">
        <wsdl:part element="impl:selectProjectByNameResponse"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="freezeLabelRequest">
        <wsdl:part element="impl:freezeLabel"
name="parameters"/>

```



```

</wsdl:message>
<wsdl:message name="getLatestVersionResponse">
  <wsdl:part element="impl:getLatestVersionResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createLabelForFileResponse">
  <wsdl:part element="impl:createLabelForFileResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createLabelResponse">
  <wsdl:part element="impl:createLabelResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createLinkResponse">
  <wsdl:part element="impl:createLinkResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createLabelForFileRequest">
  <wsdl:part element="impl:createLabelForFile"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createProjectRequest">
  <wsdl:part element="impl:createProject"
name="parameters"/>
</wsdl:message>
<wsdl:message name="setCRStatusRequest">
  <wsdl:part element="impl:setCRStatus"
name="parameters"/>
</wsdl:message>
<wsdl:message name="loginResponse">
  <wsdl:part element="impl:loginResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createProjectResponse">
  <wsdl:part element="impl:createProjectResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="selectProjectByNameRequest">
  <wsdl:part element="impl:selectProjectByName"
name="parameters"/>
</wsdl:message>
<wsdl:portType name="StarTeamWrapper">
  <wsdl:operation name="login">
    <wsdl:input message="impl:loginRequest"
name="loginRequest"/>
    <wsdl:output message="impl:loginResponse"
name="loginResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getProjects">
    <wsdl:input message="impl:getProjectsRequest"
name="getProjectsRequest"/>
    <wsdl:output message="impl:getProjectsResponse"

```

```

name="getProjectsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="createLink">
    <wsdl:input message="impl:createLinkRequest"
name="createLinkRequest"/>
    <wsdl:output message="impl:createLinkResponse"
name="createLinkResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getLatestVersion">
    <wsdl:input message="impl:getLatestVersionRequest"
name="getLatestVersionRequest"/>
    <wsdl:output message="impl:getLatestVersionResponse"
name="getLatestVersionResponse"/>
  </wsdl:operation>
  <wsdl:operation name="createTask">
    <wsdl:input message="impl:createTaskRequest"
name="createTaskRequest"/>
    <wsdl:output message="impl:createTaskResponse"
name="createTaskResponse"/>
  </wsdl:operation>
  <wsdl:operation name="createLabel">
    <wsdl:input message="impl:createLabelRequest"
name="createLabelRequest"/>
    <wsdl:output message="impl:createLabelResponse"
name="createLabelResponse"/>
  </wsdl:operation>
  <wsdl:operation name="createProject">
    <wsdl:input message="impl:createProjectRequest"
name="createProjectRequest"/>
    <wsdl:output message="impl:createProjectResponse"
name="createProjectResponse"/>
  </wsdl:operation>
  <wsdl:operation name="selectProjectByName">
    <wsdl:input message="impl:selectProjectByNameRequest"
name="selectProjectByNameRequest"/>
    <wsdl:output
message="impl:selectProjectByNameResponse"
name="selectProjectByNameResponse"/>
  </wsdl:operation>

  <wsdl:operation name="updateFile">
    <wsdl:input message="impl:updateFileRequest"
name="updateFileRequest"/>
    <wsdl:output message="impl:updateFileResponse"
name="updateFileResponse"/>
  </wsdl:operation>
  <wsdl:operation name="createLabelForFile">
    <wsdl:input message="impl:createLabelForFileRequest"

```

```

name="createLabelForFileRequest"/>
  <wsdl:output
message="impl:createLabelForFileResponse"
name="createLabelForFileResponse"/>
  </wsdl:operation>
  <wsdl:operation name="freezeLabel">
    <wsdl:input message="impl:freezeLabelRequest"
name="freezeLabelRequest"/>
    <wsdl:output message="impl:freezeLabelResponse"
name="freezeLabelResponse"/>
  </wsdl:operation>
  <wsdl:operation name="getLinkedItems">
    <wsdl:input message="impl:getLinkedItemsRequest"
name="getLinkedItemsRequest"/>
    <wsdl:output message="impl:getLinkedItemsResponse"
name="getLinkedItemsResponse"/>
  </wsdl:operation>
  <wsdl:operation name="setCRStatus">
    <wsdl:input message="impl:setCRStatusRequest"
name="setCRStatusRequest"/>
    <wsdl:output message="impl:setCRStatusResponse"
name="setCRStatusResponse"/>
  </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="StarTeamWrapperSoapBinding"
type="impl:StarTeamWrapper">
  <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="login">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="loginRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="loginResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="getProjects">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getProjectsRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getProjectsResponse">
      <wsdlsoap:body use="literal"/>
    </wsdl:output>
  </wsdl:operation>
  <wsdl:operation name="createLink">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="createLinkRequest">
      <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="createLinkResponse">

```

```

        <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="getLatestVersion">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getLatestVersionRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="getLatestVersionResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="createTask">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="createTaskRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="createTaskResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="createLabel">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="createLabelRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="createLabelResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="createProject">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="createProjectRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="createProjectResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="selectProjectByName">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="selectProjectByNameRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="selectProjectByNameResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>
<wsdl:operation name="updateFile">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="updateFileRequest">
        <wsdlsoap:body use="literal"/>
    </wsdl:input>
    <wsdl:output name="updateFileResponse">
        <wsdlsoap:body use="literal"/>
    </wsdl:output>
</wsdl:operation>

```

```

        </wsdl:input>
        <wsdl:output name="updateFileResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="createLabelForFile">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="createLabelForFileRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="createLabelForFileResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="freezeLabel">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="freezeLabelRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="freezeLabelResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="getLinkedItems">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="getLinkedItemsRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="getLinkedItemsResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="setCRStatus">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="setCRStatusRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="setCRStatusResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="StarTeamWrapperService">
    <wsdl:port binding="impl:StarTeamWrapperSoapBinding"
name="StarTeamWrapper">
        <wsdlsoap:address
location="http://localhost:8080/BorlandStarTeamWS/services/StarTeamWrapper"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

## eMailSystemWrapper WSDL

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://email.ws.tez.alpay.erturkmen.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://email.ws.tez.alpay.erturkmen.com"
xmlns:intf="http://email.ws.tez.alpay.erturkmen.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
<!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
  <wsdl:types>
    <schema elementFormDefault="qualified"
targetNamespace="http://email.ws.tez.alpay.erturkmen.com"
xmlns="http://www.w3.org/2001/XMLSchema">
      <element name="send">
        <complexType>
          <sequence>
            <element name="toAddress" type="xsd:string"/>
            <element name="mailSubject" type="xsd:string"/>
            <element name="mailBody" type="xsd:string"/>
            <element name="attachment" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
      <element name="sendResponse">
        <complexType>
          <sequence>
            <element name="sendReturn" type="xsd:string"/>
          </sequence>
        </complexType>
      </element>
    </schema>
  </wsdl:types>
  <wsdl:message name="sendResponse">
    <wsdl:part element="impl:sendResponse"
name="parameters"/>
  </wsdl:message>
  <wsdl:message name="sendRequest">
    <wsdl:part element="impl:send" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="EmailSystemWrapper">
    <wsdl:operation name="send">
      <wsdl:input message="impl:sendRequest"
name="sendRequest"/>
      <wsdl:output message="impl:sendResponse"
name="sendResponse"/>
    </wsdl:operation>
  </wsdl:portType>
</wsdl:definitions>
```

```

        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="EmailSystemWrapperSoapBinding"
type="impl:EmailSystemWrapper">
        <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="send">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="sendRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="sendResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="EmailSystemWrapperService">
        <wsdl:port binding="impl:EmailSystemWrapperSoapBinding"
name="EmailSystemWrapper">
            <wsdlsoap:address
location="http://localhost:8081/EmailSystemWS/services/EmailSy
stemWrapper"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>

```

## FileSystemWrapper WSDL

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://file.ws.tez.alpay.erturkmen.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://file.ws.tez.alpay.erturkmen.com"
xmlns:intf="http://file.ws.tez.alpay.erturkmen.com"
xmlns:wSDL="http://schemas.xmlsoap.org/wSDL/"
xmlns:wSDLsoap="http://schemas.xmlsoap.org/wSDL/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
    <wsdl:types>
        <schema elementFormDefault="qualified"
targetNamespace="http://file.ws.tez.alpay.erturkmen.com"
xmlns="http://www.w3.org/2001/XMLSchema">
            <element name="install">
                <complexType/>
            </element>
            <element name="installResponse">
                <complexType/>
            </element>

```

```

<element name="createFolder">
  <complexType>
    <sequence>
      <element name="parentFolderPath" type="xsd:string"/>
      <element name="folderName" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="createFolderResponse">
  <complexType>
    <sequence>
      <element name="createFolderReturn" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="generateDocument">
  <complexType>
    <sequence>
      <element name="iniFile" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="generateDocumentResponse">
  <complexType>
    <sequence>
      <element name="generateDocumentReturn"
type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="appendFile">
  <complexType>
    <sequence>
      <element name="siiFile" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="appendFileResponse">
  <complexType>
    <sequence>
      <element name="appendFileReturn" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="renameFolder">
  <complexType>
    <sequence>
      <element name="parentFolderPath" type="xsd:string"/>
      <element name="oldName" type="xsd:string"/>
      <element name="newName" type="xsd:string"/>
    </sequence>
  </complexType>

```



```

</element>
<element name="renameFolderResponse">
  <complexType>
    <sequence>
      <element name="renameFolderReturn" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="extractPackage">
  <complexType>
    <sequence>
      <element name="parentFolderPath" type="xsd:string"/>
      <element name="packageName" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
<element name="extractPackageResponse">
  <complexType>
    <sequence>
      <element name="extractPackageReturn" type="xsd:string"/>
    </sequence>
  </complexType>
</element>
</schema>
</wsdl:types>
<wsdl:message name="renameFolderResponse">
  <wsdl:part element="impl:renameFolderResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="generateDocumentResponse">
  <wsdl:part element="impl:generateDocumentResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="createFolderResponse">
  <wsdl:part element="impl:createFolderResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="installResponse">
  <wsdl:part element="impl:installResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="extractPackageResponse">
  <wsdl:part element="impl:extractPackageResponse"
name="parameters"/>
</wsdl:message>
<wsdl:message name="renameFolderRequest">
  <wsdl:part element="impl:renameFolder"
name="parameters"/>
</wsdl:message>
<wsdl:message name="generateDocumentRequest">
  <wsdl:part element="impl:generateDocument"
name="parameters"/>

```

```

    </wsdl:message>
    <wsdl:message name="createFolderRequest">
      <wsdl:part element="impl:createFolder"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="appendFileRequest">
      <wsdl:part element="impl:appendFile" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="extractPackageRequest">
      <wsdl:part element="impl:extractPackage"
name="parameters"/>
    </wsdl:message>
    <wsdl:message name="appendFileResponse">
      <wsdl:part element="impl:appendFileResponse"
name="parameters"/>
    </wsdl:message>
  </wsdl:message>
<wsdl:message name="installRequest">
  <wsdl:part element="impl:install" name="parameters"/>
</wsdl:message>
<wsdl:portType name="FileSystemWrapper">
  <wsdl:operation name="install">
    <wsdl:input message="impl:installRequest"
name="installRequest"/>
    <wsdl:output message="impl:installResponse"
name="installResponse"/>
  </wsdl:operation>
  <wsdl:operation name="createFolder">
    <wsdl:input message="impl:createFolderRequest"
name="createFolderRequest"/>
    <wsdl:output message="impl:createFolderResponse"
name="createFolderResponse"/>
  </wsdl:operation>
  <wsdl:operation name="generateDocument">
    <wsdl:input message="impl:generateDocumentRequest"
name="generateDocumentRequest"/>
    <wsdl:output message="impl:generateDocumentResponse"
name="generateDocumentResponse"/>
  </wsdl:operation>
  <wsdl:operation name="appendFile">
    <wsdl:input message="impl:appendFileRequest"
name="appendFileRequest"/>
    <wsdl:output message="impl:appendFileResponse"
name="appendFileResponse"/>
  </wsdl:operation>
  <wsdl:operation name="renameFolder">
    <wsdl:input message="impl:renameFolderRequest"
name="renameFolderRequest"/>
    <wsdl:output message="impl:renameFolderResponse"
name="renameFolderResponse"/>
  </wsdl:operation>
  <wsdl:operation name="extractPackage">

```

```

        <wsdl:input message="impl:extractPackageRequest"
name="extractPackageRequest"/>
        <wsdl:output message="impl:extractPackageResponse"
name="extractPackageResponse"/>
    </wsdl:operation>
</wsdl:portType>
<wsdl:binding name="FileSystemWrapperSoapBinding"
type="impl:FileSystemWrapper">
    <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="install">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="installRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="installResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="createFolder">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="createFolderRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="createFolderResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="generateDocument">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="generateDocumentRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="generateDocumentResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="appendFile">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="appendFileRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="appendFileResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="renameFolder">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="renameFolderRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>

```

```

        <wsdl:output name="renameFolderResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
    <wsdl:operation name="extractPackage">
        <wsdlsoap:operation soapAction=""/>
        <wsdl:input name="extractPackageRequest">
            <wsdlsoap:body use="literal"/>
        </wsdl:input>
        <wsdl:output name="extractPackageResponse">
            <wsdlsoap:body use="literal"/>
        </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
<wsdl:service name="FileSystemWrapperService">
    <wsdl:port binding="impl:FileSystemWrapperSoapBinding"
name="FileSystemWrapper">
        <wsdlsoap:address
location="http://localhost:8081/FileSystemWS/services/FileSyst
emWrapper"/>
    </wsdl:port>
</wsdl:service>
</wsdl:definitions>

```

## ProjectRepositoryWrapper WSDL

```

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
targetNamespace="http://projectrepo.tez.alpay.erturkmen.com"
xmlns:apachesoap="http://xml.apache.org/xml-soap"
xmlns:impl="http://projectrepo.tez.alpay.erturkmen.com"
xmlns:intf="http://projectrepo.tez.alpay.erturkmen.com"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema">
    <!--WSDL created by Apache Axis version: 1.4
Built on Apr 22, 2006 (06:55:48 PDT)-->
    <wsdl:types>
        <schema elementFormDefault="qualified"
targetNamespace="http://projectrepo.tez.alpay.erturkmen.com"
xmlns="http://www.w3.org/2001/XMLSchema">
            <element name="lookup">
                <complexType>
                    <sequence>
                        <element name="key" type="xsd:string"/>
                    </sequence>
                </complexType>
            </element>
            <element name="lookupResponse">
                <complexType>

```

```

        <sequence>
            <element name="lookupReturn" type="xsd:string"/>
        </sequence>
    </complexType>
</element>
</schema>
</wsdl:types>
    <wsdl:message name="lookupRequest">
        <wsdl:part element="impl:lookup" name="parameters"/>
    </wsdl:message>
    <wsdl:message name="lookupResponse">
        <wsdl:part element="impl:lookupResponse"
name="parameters"/>
    </wsdl:message>
    <wsdl:portType name="ProjRepoWrapper">
        <wsdl:operation name="lookup">
            <wsdl:input message="impl:lookupRequest"
name="lookupRequest"/>
            <wsdl:output message="impl:lookupResponse"
name="lookupResponse"/>
        </wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="ProjRepoWrapperSoapBinding"
type="impl:ProjRepoWrapper">

        <wsdlsoap:binding style="document"
transport="http://schemas.xmlsoap.org/soap/http"/>
        <wsdl:operation name="lookup">
            <wsdlsoap:operation soapAction=""/>
            <wsdl:input name="lookupRequest">
                <wsdlsoap:body use="literal"/>
            </wsdl:input>
            <wsdl:output name="lookupResponse">
                <wsdlsoap:body use="literal"/>
            </wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="ProjRepoWrapperService">
        <wsdl:port binding="impl:ProjRepoWrapperSoapBinding"
name="ProjRepoWrapper">
            <wsdlsoap:address
location="http://localhost:8080/ProjRepoWS/services/ProjRepoWr
apper"/>
        </wsdl:port>
    </wsdl:service>
</wsdl:definitions>

```

## APPENDIX H: APPLICATION CODE DEVELOPED FOR WEB SERVICES (CASE STUDY I)

### CaliberRMWrapper

```
package com.erturkmen.alpay.tez.ws.caliberrm;
import com.starbase.caliber.*;
import com.starbase.caliber.Baseline;
import com.starbase.caliber.BaselineTree;
import com.starbase.caliber.server.CaliberServer;
import com.starbase.caliber.server.RemoteServerException;
public class CaliberRMWrapper {
    private static Session getSession(String serverAddress,
String userName, String password) {
        try {
            return(new
CaliberServer(serverAddress)).login( userName, password);
        } catch(Exception e) {
            e.printStackTrace();
        }
        return null;
    }
    private static void logoff(Session session) {
        session.logout();
    }

    public static int selectProjectByName(String
serverAddress, String userName, String password, String
projectName) { //Sequence7
```

```

        Session session = getSession(serverAddress,
userName, password);
        Project[] projects;
        try {
            projects = session.getProjects();
            Project selectedProject = null;
            selectedProject = null;
            for(int projectID=0; projectID<
projects.length; projectID++) {
                selectedProject = projects[projectID];
                if(selectedProject.getName().equals(projectName))
break;
            }
            logoff(session);
            return
selectedProject.getProjectID().getIDNumber();
        } catch (RemoteServerException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
            logoff(session);
            return -1;
        }
    }
    private static Project selectProjectByID(String
serverAddress, String userName, String password, int
projectID) {
        Session session = getSession(serverAddress,
userName, password);
        try {
            return (Project) session.get(new
ProjectID(projectID));
        } catch (RemoteServerException e1) {
            // TODO Auto-generated catch block
            e1.printStackTrace();
            return null;
        }
    }
    public static int createBaseline(String serverAddress,
String userName, String password, int projectID, String name)
{ //Sequence7
        try {
            Session session = getSession(serverAddress,
userName, password);
            Baseline baseline = new Baseline(name, new
ProjectID(projectID), session);
            baseline.save();
            System.out.println(selectProjectByID(serverAddress,
userName, password,
projectID).getCurrentBaseline().getRequirementTree().getRoot()
.getChildCount());
            BaselineTree tree = new
BaselineTree(baseline, selectProjectByID(serverAddress,

```

```

userName,                                     password,
projectID).getCurrentBaseline().getRequirementTree(),
session);
        tree.save();

        System.out.println(baseline.getRequirementTree().toString());
        baseline.save();
        return
baseline.getBaselineID().getIDNumber();
    } catch (Exception e) {
        e.printStackTrace();
        return -1;
    }
}
public static int createProject(String serverAddress,
String userName, String password, String projectName) {
    Session session = getSession(serverAddress,
userName, password);
    try {
        Project project = new Project(projectName,
session);
        project.save();
        return project.getID().getIDNumber();
    } catch (RemoteServerException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return -1;
    }
}
}
}

```

## StarTeamWrapper

```

package com.erturkmen.alpay.tez.ws.starteam;

import java.text.DateFormat;
import java.text.SimpleDateFormat;
import java.util.Date;

import com.borland.starteam.impl.Folder;
import com.borland.starteam.impl.Item;
import com.borland.starteam.impl.Link;
import com.borland.starteam.impl.StarTeamFinder;
import com.borland.starteam.impl.Task;
import com.borland.starteam.impl.util.DateTime;
import com.borland.starteam.impl.View;
import com.borland.starteam.impl.ChangeRequest;

import com.borland.starteam.impl.Label;

```



```

import com.borland.starteam.impl.LinkCache;
import com.starbase.starteam.Project;
import com.starbase.starteam.Server;
import com.starbase.starteam.StarTeamURL;

public class StarTeamWrapper {
    private static Server login(String serverAddress, String
userName, String password) {
        Server server = new Server(serverAddress, 49201);
        try {
            server.logOn(userName, password);
        } catch (Exception e) {
            e.printStackTrace();
            System.out.println("Can not connect to
server");
            return null;
        }
        return server;
    }
    public static String login(String url) { //Sequence7
        StarTeamURL stUrl = getStarTeamURL(url);
        Server server = login(stUrl.getHostName(),
stUrl.getUserName(), stUrl.getPassword());
        String returnString = "connected to: " +
server.toString();
        server.disconnect();
        return returnString;
    }
    private static Server login(StarTeamURL stUrl) {
        return login(stUrl.getHostName(),
stUrl.getUserName(), stUrl.getPassword());
    }
    public static String selectProjectByName(String url,
String projectName) {
        StarTeamURL stUrl = getStarTeamURL(url);
        Server server = login(stUrl.getHostName(),
stUrl.getUserName(), stUrl.getPassword());
        StarTeamFinder.openProject(url);
        Project[] projects = server.getProjects();
        Project selectedProject = null;
        for(int projectID=0; projectID< projects.length;
projectID++) {
            selectedProject = projects[projectID];

            if(selectedProject.getName().equals(projectName)) break;
        }
        server.disconnect();
        return url+selectedProject.getName()+"/";
    }
    public static String updateFile(String url, String
filename) { //Sequence7

```

```

        Folder folder = StarTeamFinder.openFolder(url);
        com.borland.starteam.impl.File file =
StarTeamFinder.findFile(folder, filename, false);
        try {
            file.checkin();
            return "File checked-in
from:"+file.getLocalPath()+file.getLocalName();
        } catch (Exception e) {
            e.printStackTrace();
            return "Check-in failed!!!";
        }
    }
    public static String getLatestVersion(String url) {
        Folder folder = StarTeamFinder.openFolder(url);
        // StarTeamFinder.
        com.borland.starteam.impl.File file =
StarTeamFinder.findFile(folder, filename, false);
        try {
            file.checkout();
            return "File checked-out
to:"+file.getLocalPath()+file.getLocalName();
        } catch (Exception e) {
            e.printStackTrace();
            return "Check-out failed!!!";
        }
    }

    public static int createLabelForFile(String url, String
filename) { //Sequence7
        com.borland.starteam.impl.View view =
StarTeamFinder.openView(url);
        com.borland.starteam.impl.Label label =
view.createViewLabel(filename+" "+getDateTIme(), "Created
after checking-in: "+filename, DateTime.now(), false, true);
        return label.getID();
    }
    public static int createLabel(String url, String
labelBaseName) { //Sequence3
        com.borland.starteam.impl.View view =
StarTeamFinder.openView(url);
        com.borland.starteam.impl.Label label =
view.createViewLabel(labelBaseName+" "+getDateTIme(), "Created
on: "+DateTime.now(), DateTime.now(), false, true);
        return label.getID();
    }
    public static String freezeLabel(String url, int
labelID) { //Sequence3
        com.borland.starteam.impl.View view =
StarTeamFinder.openView(url);
        Label label = findLabelByID(view, labelID);
        label.setLocked(true);
        return "Successful!";
    }

```

```

    }
    private static Label findLabelByID(View view, int
labelID) {
        // TODO Auto-generated method stub
        com.borland.starteam.impl.Label[] labels =
view.getLabels();
        com.borland.starteam.impl.Label selectedLabel =
null;
        for (int i=0; i<labels.length; i++) {
            selectedLabel=labels[i];
            if (labelID==selectedLabel.getID()) break;
        }
        return selectedLabel;
    }
    private static String getDateTIme() {
        DateFormat dateFormat = new
SimpleDateFormat("yyyy/MM/dd-HH:mm:ss");
        Date date = new Date();
        return dateFormat.format(date);
    }
    private static StarTeamURL getStarTeamURL(String url) {
        return new StarTeamURL(url);
    }
    public static String[] getProjects(String url) {
//Sequence7_Form
        StarTeamURL stUrl = getStarTeamURL(url);
        Server server = login(stUrl.getHostName(),
stUrl.getUserName(), stUrl.getPassword());
        Project[] projects = server.getProjects();
        String[] projectInfo = new String[projects.length];
        for(int i=0; i<projects.length; i++) {
            projectInfo[i] = projects[i].getName();
        }
        return projectInfo;
    }
    public static String createProject(String serverAddress,
String username, String password, String projectName, String
workingFolderPath) {
        Server server = login(serverAddress, username,
password);
        Project project = new Project(server, projectName,
workingFolderPath);
        return
("starteam://" +username+":"+password+"@"+serverAddress+":49201
/"+projectName);
    }

    public static int createTask(String url, String name) {
        Folder folder = StarTeamFinder.openFolder(url);
        Task task = new Task(folder);
        task.setName(name);
        return task.getID();
    }

```

```

    }
    public static int createLink(String url1, int item1ID,
String item1Type, String url2, int item2ID, String item2Type)
{
    Link link = new Link(getItem(url1, item1ID,
item1Type), getItem(url2, item2ID, item2Type));
    return link.getID();
}
    public static int[] getLinkedItems(String url, int
itemID, String itemType) {
    Item item = getItem(url, itemID, itemType);
    View view = StarTeamFinder.openView(url);
    LinkCache linkCache = new LinkCache();
    Link[] links = linkCache.getLinks(item);
    Item[] linkedItems = new Item[links.length];
    int[] linkedItemIDs = new int[links.length];
    for (int i=0; i<links.length; i++) {
        linkedItems[i] = (Item)
links[i].resolveChild();
        if (linkedItems[i].getType() ==
view.getServer().typeForName("TASK")) linkedItemIDs[i] =
linkedItems[i].getID();
    }
    return linkedItemIDs;
}
    private static Item getItem(String url, int itemID,
String itemType) {
    View view = StarTeamFinder.openView(url);
    return
view.findItem(view.getServer().typeForName(itemType), itemID);
}
    public static void setCRStatus(String url, int[]
itemIDs, int status) {
    ChangeRequest cr = null;
    for (int i=0; i<itemIDs.length; i++) {
        cr = ((ChangeRequest) (getItem(url,
itemIDs[i], "CHANGEREQUEST")));
        cr.setStatus(status);
    }
}
}
}

```

## eMailWrapper

```

package com.erturkmen.alpay.tez.ws.email;

import javax.mail.*;
import javax.mail.internet.*;

import java.util.Properties;

```

```

import javax.activation.FileDataSource;
import javax.activation.DataHandler;

public class EmailSystemWrapper {
    public static String send(String toAddress, String
mailSubject, String mailBody, String attachment) throws
Exception {
        Properties props = new Properties();
        props.put("mail.smtps.auth", "true");
        Session session = Session.getDefaultInstance(props,
null);
        MimeMessage msg = new MimeMessage(session);
        msg.setSubject(mailSubject);
        msg.setContent(mailBody, "text/html");
        /* Attachments not supported
        *      MimeBodyPart      attachFilePart      =      new
MimeBodyPart();
        FileDataSource fds =
            new FileDataSource(attachment);
        attachFilePart.setDataHandler(new
DataHandler(fds));
        attachFilePart.setFileName(fds.getName());
        Multipart mp = new MimeMultipart();
        mp.addBodyPart(textPart);
        mp.addBodyPart(attachFilePart);
        message.setContent(mp);*/
        msg.setFrom(new
InternetAddress("alpaye@gmail.com"));
        msg.addRecipient(Message.RecipientType.TO,      new
InternetAddress(toAddress));
        Transport t = session.getTransport("smtps");
        t.connect("smtp.gmail.com",      "alpaye@gmail.com",
        "");
        t.sendMessage(msg, msg.getAllRecipients());
        return "success";
    }
}

```

## **FileWrapper**

```

package com.erturkmen.alpay.tez.ws.file;
import java.io.File;
import java.io.IOException;

public class FileSystemWrapper {
    private static String execute(String path, String
statement) {
        try {
            File dir = new File(path);

```

```

        Process p =
Runtime.getRuntime().exec("c:\\windows\\system32\\cmd.exe /c
"+statement, null, dir);
        int exitVal = p.waitFor();
        return (new Integer(exitVal)).toString();
    } catch (IOException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        return "failure";
    } catch (Exception e) {
        e.printStackTrace();
        return "unknown error";
    }
}

public static String generateDocument(String iniFile) {
    //expects e.g.
iniFile=CustomerReqDocConf_RE5213.ini
    String statement = "docfactory -autofile
"C:\\dev\\pletin\\conf\\docFac\\"+iniFile+"\"";
    System.out.println("statement:"+statement);
    return execute("C:\\Program
Files\\Borland\\CaliberRM\\", statement);
}

public static String appendFile(String siiFile) {
    //expects e.g. siiFile=SRSDocConf_RE52212.SII
    //converts all docx files in C:\\dev\\docMerge\\,
converts them to doc, and appends them into out.doc

    String statement = "ConvertDoc.exe
/J\"C:\\dev\\pletin\\conf\\docMerge\\"+siiFile+"\"";
    //String statement = "docMerge.bat";
    System.out.println(statement);
    System.out.println(execute("C:\\Program
Files\\Softinterface, Inc\\Convert Doc\\", statement));
    execute("C:\\dev\\docMerge\\", "ren out.DOC
SRS.DOC");
    return execute("C:\\dev\\docMerge\\", "xcopy
SRS.DOC \"C:\\dev\\pletin\\Project Documents\\" /Y");
}

public static String createFolder(String
parentFolderPath, String folderName) {
    String statement = "mkdir "+folderName;
    return execute(parentFolderPath, statement);
}

public static String renameFolder(String
parentFolderPath, String oldName, String newName) {
    String statement = "ren "+oldName+" "+newName;
    return execute(parentFolderPath, statement);
}

public static String extractPackage(String
parentFolderPath, String packageName) {

```

```

        String statement = "unzip "+packageName;
        return execute(parentFolderPath, statement);
    }
}

```

## ProjectRepository Wrapper

```

package com.erturkmen.alpay.tez.projectrepo;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class ProjRepoWrapper {
    public static String lookup(String key) {
        // TODO Auto-generated method stub
        Connection conn = null;
        Statement stmt = null;
        ResultSet rs = null;
        String value = null;
        try {
            conn =
DriverManager.getConnection("jdbc:mysql://192.168.74.131/pleti
n?" +
                                "user=root&password=");
            stmt = conn.createStatement();
            rs = stmt.executeQuery("SELECT * FROM
`ProjectInfo` WHERE (`Key` = \""+key+"\")");
            rs.first();
            value = rs.getString("Value");
        } catch (SQLException ex) {
            ex.printStackTrace();
        } finally {
            // it is a good idea to release
            // resources in a finally{} block
            // in reverse-order of their creation
            // if they are no-longer needed
            if (rs != null) {
                try {
                    rs.close();
                } catch (SQLException sqlEx) { }
                rs = null;
            }
            if (stmt != null) {
                try {
                    stmt.close();
                } catch (SQLException sqlEx) { }
                stmt = null;
            }
        }
        return value;    }}

```

# VITA

## PERSONAL INFORMATION

Surname, Name: Ertürkmen, Kulubey Alpay  
Nationality: Turkish (TR)  
Date and Place of Birth: March 24, 1981, Ankara  
Marital Status: Married  
Phone: +90 533 6324473  
e-Mail: [alpaye@gmail.com](mailto:alpaye@gmail.com)

## EDUCATION

Degree	Institution	Year of Graduation
MS	METU, Informatics Institute	2003
BS	METU, Industrial Engineering	2001
High School	Özel Aykan Koleji, Ankara	1997

## WORK EXPERIENCE

Year	Place	Enrollment
2004-2009	Bilgi ve Teknoloji Grubu	Sr. Technical Consultant
2001-2003	METU, Informatics Institute	Research Assistant

## FOREIGN LANGUAGES

English (Advanced)

## PUBLICATIONS

1. Erturkmen, K.A., Demirors, O. (2009). Integration of CASE Tools to Software Processes: A Case Study. In Industrial Proceedings of 16th European Systems and Software Process Improvement and Innovation Conference (EuroSPI'2009) (pp:11.1-11.2).