Technische Dokumentation für die Palladio-Erweiterung IntBIIS\_LP



FZI Forschungszentrum Informatik

Version 1.0

Inhaltsverzeichnis

[1 Übersicht 3](#_Toc528068687)

[1.1 Anforderungen 3](#_Toc528068688)

[1.2 Vorstellung enthaltener Konzepte 3](#_Toc528068689)

[1.3 Unterschiede des Systems 3](#_Toc528068690)

[2 Erstellte Komponenten und Elemente 4](#_Toc528068691)

[2.1 Grundlagen der Prozessmodellierung mit Palladio, IntBIIS und IntBIIS\_LP 4](#_Toc528068692)

[2.2 Ursprünglich für IntBIIS erstellte Diagramme 4](#_Toc528068693)

[2.3 Für IntBIIS\_LP erstellte Diagramme, Elemente und Komponenten 4](#_Toc528068694)

[2.3.1 Fallstudien 5](#_Toc528068695)

[2.3.2 IntBIIS\_LP-Diagramme 5](#_Toc528068696)

[2.3.3 Erweiterungen des Repositories 5](#_Toc528068697)

[2.3.3.1 Ergänzte Schnittstellen 5](#_Toc528068698)

[2.3.3.2 Ergänzte Interfaces: 6](#_Toc528068699)

[2.3.3.3 Datenobjekte im Repository 6](#_Toc528068700)

[2.3.3.4 Datenobjekte im Datamodel 7](#_Toc528068701)

[2.3.3.5 Änderungen an UseCases: 7](#_Toc528068702)

[3 Generelle Probleme und Anmerkungen 8](#_Toc528068703)

[4 Verwendete Ressourcen 9](#_Toc528068704)

# Übersicht

In diesem Dokument wird der Fortschritt bei der Einarbeitung der Fallstudien 9, 10 und 13 festgehalten. Dreh und Angel-Punkt sind dabei das Programm Palladio, dessen Erweiterung IntBIIS und IntBIIS\_LP. Es richtet sich an FZI-Mitarbeiter welche künftig an dieser Stelle ansetzen und die Arbeit fortführen bzw. beenden werden.

Der Fokus liegt dabei auf den schon eingebauten Änderungen und Erkenntnissen, die bei der Erweiterung gewonnen wurden.

## Anforderungen

Folgende Anforderungen wurden zu beginnen der Fallstudieneinarbeitung gestellt:

* Vorhandene Fallstudien müssen kopiert werden, falls eine Erweiterung oder Anpassung nötig wird
* Alle Änderungen müssen dokumentiert werden
* Die erweiternden Diagramme von IntBIIS\_LP müssen zu jedem vorhandenen und jedem weiteren Fall modelliert werden

## Vorstellung enthaltener Konzepte

m die Arbeit an den Fallstudien und der dafür nötigen Erweiterung der Komponenten, Datenobjekte, Interfaces etc. durchführen zu können, muss ein gewisses Grundwissen über den Aufbau von Palladio und den genannten Erweiterungen vorhanden sein. Dieses Wissen kann gewonnen werden, indem die entsprechende Dokumentation gelesen wird. Empfohlen wird darüber hinaus, Kontakt zu einem erfahrenen Kollegen zu suchen um Unklarheiten zu beseitigen, da die Komplexität nicht trivial ist.

Es wird also ein umfassendes Grundwissen vorausgesetzt und somit nur die Änderungen und keine Konzepte erklärt.

Alle modell-arten als Liste und eine kurze Erklärung, was was enthält IN TABELLENFORM

|  |  |
| --- | --- |
| Modell | Erklärung |
| .bpusagemodel | Enthält die Prozesslogik |
| .allocation |  |
| .composite\_diagramm | Enthält ein Diagramm von einer .system-Datei |
| .datamodel | Enthält die Datenobjekte für die prozessorientierte Seite palladios |
| .organizationenvironmentmodel | Enthält die Elemente die zu der Organisation gehören. Also geräte und Rollen |
| .repository | Ist der große Speicher unter dem alles was für das Modellierungsprojekt notwenig ist. Ist mit einem Repository von git zu vergleichen. Gehört eher zur Technischen Seite |
| .resourceenvironment | Enthält detailierte techinsche Elemente des Projektes. Beispielweise die Repräsentation von StoreServer oder CPU und den Primitiven Datentypen |
| .system | Enthält die Kompletten Beziehungen zwischen den Komponenten |

## Unterschiede des Systems

Das CoCoMe-Kassensystem welches als Grundlage zum Modellieren des Systems dient, ist ein Kassensystem für eine Einzelhandelskette und wird zum Teil in der Realität tatsächlich verwendet. Allerdings wird es auch von der Uni Karlsruhe ständig weiterentwickelt. Alle verwendeten Fallstudien sind noch im Ursprünglichen Konzept des Systems enthalten, auch wenn sich die Gründe für dessen Implementierung verändert haben. Das bedeutet, es wurden beim Entwurf des CoCoMe-Kassensystems schon unterschiedliche Ausbaustufen erdacht und deren Konzepte in schriftlicher Form festgehalten. Diese Dokumentation bezieht sich hauptsächlich auf die letzte Ausbaustufe, welche das Integrieren eine sogenannten PickUp-Shops-Services in den Web-Store vorsieht. Dadurch wird es den Kunden ermöglicht, Produkte Online zu bestellen und diese in einer Filliale ihrer Wahl abholen zu können. In den Dokumentationen zu CoCoMe wird näher auf diese Unterschiedlichen Ausbaustufen eingegangen und es stehen Komponenten-Diagramme für jede Ausbaustufe bereit.

Die Fallstudien 09 – 13 können der letzten und Größten (die mit dem PickUp-Shop Figure: 5.11 in der Cocome-Dokumentation) Ausbaustufe zugeordnet werden, da Teile der Componente PickupShop integriert werden mussten. Ob oder wie die vorherigen Fallstudien unterteilt und Zugeordnet werden können, entzieht sich der Kenntnis des Autors.

# Erstellte Komponenten und Elemente

Um die bisherigen Erweiterungen zu erhalten, muss das Projekt aus dem ESS-Gitlab geladen werden. Dort sind die Unveränderten sowie die angepassten Fallstudien (auch UseCase genannt) 1-8 enthalten, sowie die neu erstellten Fallstudien 9,10 und 13. Diese sollten sich ausschließlich darin unterscheiden, dass sie von zwei unterschiedlichen Personen erstellt wurden. Das bedeutet, dass das gleiche Maß an Sorgfalt angelegt wurde, jedoch bei Prozessen immer Subjektive unterschiede entstehen können. Dies resultiert aus dem Verständnis des beschriebenen Falls und den eigenen Vorstellungen wie das fertige System tatsächlich aussehen kann/soll. Denn die beiden Ersteller hatten weder Räumlich noch zeitlich zusammen an den Prozessen gearbeitet.

Für jeden dieser UseCases gibt es ein Usagemodel und die darin benötigten Komponenten, Interfaces, Datentypen und sonstige Elemente. Sie alle sind in einem Repository enthalten und stehen in Beziehung miteinander. Deswegen ist es nur bedingt ratsam Teile davon umzubenennen, denn es könnte passieren, dass die Beziehungen dadurch zerstört werden und funktionieren Diagramme zu fehlerhaften werden.

## Grundlagen der Prozessmodellierung mit Palladio, IntBIIS und IntBIIS\_LP

Einer der wichtigsten Modellierungsaspekte ist: die in den Sequenzdiagrammen aufgezeigten Abläufe sind in den Usagemodellen nur bis zur Ersten Komponente/Interface relevant für das USageModel. Denn die erste Komponente wird mehr oder weniger direkt vom Nutzer angesprochen, alle weiteren werden durch die Systemlogik ausgelöst.

Zum Beispiel: durch das klicken eines Buttons wird eine Funktion zur Rechnungserstellung ausgelöst. Wenn diese Funktion wiederum andere Funktionen aufruft und dadurch im Hintergrund eine weit verschachtelte Funktionsabfolge abläuft, ist der Nutzer zwar schuld daran, aber er hat dennoch nur eine Funktion angesprochen. Also sozusagen die Startfunktion. Somit braucht er nur die Berechtigung auf diese eine Funktion aber nicht auf alle anderen.

Darüber hinaus ist zu beachten, dass nur die Funktionen genutzt werden können, welche durch die Schnittstellen „außen“ am System zu Verfügung gestellt werden.

## Bereits vorhandene Diagramme für IntBIIS

UseCase 1: Process Sale  
UseCase 2: Manage Express Checkout  
UseCase 3: Order Products  
UseCase 4: Receive Ordered Products  
UseCase 5: Show Stock Reports  
UseCase 6: Show Delivery Reports  
UseCase 7: Change Price  
UseCase 8: Product Exchange (on low stock) Among Stores

## Für IntBIIS\_LP erstellte Diagramme, Elemente und Komponenten

In diesem Abschnitt ist der Kern dieser Dokumentation zu finden, denn hier wird darauf eingegangen, welche Elemente wo erstellt wurden. Außerdem auch welche Veränderungen an welchen UseCases vorgenommen wurden.

### Fallstudien

UseCase 9: Process Online Sale  
UseCase 10: Manage Product Information  
UseCase 13: View Customer Report

### IntBIIS\_LP-Diagramme

Zu jedem UseCase wurden jeweils ein RolesToPools.intbiis\_lp und ein RoleToAction.intbiis\_lp erstellt. Darin sind die für IntBIIS\_LP typischen zusätzlichen Informationen enthalten, welche ein genaueres Verständnis von zugehörigenkeiten ermöglichen.

So enthält RoleToAction.intbiis\_lp die Zuordnung von EntryLevelSystemCalls, Aquire- und ReleaseDeviceRessource zur ausführenden bzw. verantwortlichen Rolle. Wohingegen RolesToPools.intbiis\_lp die Zugehörigkeiten der einzelnen Rollen zu sogenannten OrganisationalUnits erfässt.

### Erweiterungen des Repositories

In diesem Abschnitt sind alle Anpassungen vermerkt die eingebaut wurden, um die UseCases 09, 10 und 13 möglichst korrekt und detailliert modellieren zu können. Sowie die Änderungen, die in den UseCases 01-08 gemacht wurden da sich durch Hinzufügen von Elementen detaillierte Möglichkeiten ergeben haben.

#### Ergänzte Schnittstellen

Folgende Schnittstellen wurden bestimmten Komponenten hinzugefügt:

|  |  |  |  |
| --- | --- | --- | --- |
| Ergänzte Schittstellen | | | |
| Ort: | org.cocome.cloud.web.frontend.store | | |
| Neues Element | ElementTyp | | |
| IBank | <OperationProvidedRole> | | |
| IShoppingCartView | <OperationProvidedRole> | | |
| CheckOutWizardView | <OperationProvidedRole> | | |
| Ort: | org.cocome.cloud.web.data.storedata |  |  |
| IStoreCostumerView | <OperationProvidedRole> | | |

#### Ergänzte Interfaces:

Ort: cocome-cloud.repository (runter scrollen bis ganzen großen i’s zu finden sind, allerdings ganz am Anfang dieser Liste)

Elemente:

* IBank<OperationInterface>
* IShoppingCartView<OperationInterface>
  + AddToCart()
    - Item<Parameter> -> CompositeDataType Product
  + Reset()
  + getProductList
    - return: List\_products<CollectionDataType>
* CheckOutWizardView <OperationInterface>
  + proceedCheckout <OperationInterface>
  + setCreditCard <OperationInterface>
    - CreditCard <parameter>
  + setCreditCardPin <OperationInterface>
    - creditCard
  + getCreditCard <OperationInterface>
    - return: creditCard<CompositDataType>
  + getCreditCardPin <OperationInterface>
    - return: creditCardPin<CompositDataType>
  + validateCard <OperationInterface>
    - creditCard <parameter>
    - pin <parameter>
  + debitCard <OperationInterface>
    - transactionID <parameter>
  + accountSale <OperationInterface>
    - shoppingcart <parameter>
* IStoreInformation <OperationInterface> (! Hier nur eine Funktion ergänzt)
  + setPickupStore <operationSignature>
    - storeID <Parameter>
* ICostumerView <OperationInterface>
  + getCostumerInformations <OperationSignature>
    - CustomerIdentifier<Parameter>

#### Ergänzte Äußere Interfaces

Die äußeren Schnittstellen sind die Schnittstellen, auf die der Nutzer tatsächlich zugreifen kann. Diese leiten sozusagen die Anfrage an die verknüpften Komponenten bzw. deren Interfaces weiter.

Diese findet man in .system ganz unten. Dort können sie natürlich auch mittels Tree-Editor erstellt und bearbeiten werden. Darüber hinaus können sie gesehen werden, wenn man das composite-diagramm öffnet. Falls dieses nicht vorhanden sein sollte:

Rechts-Klick auf cocome-cloud.system > Initialize a Composed Structure Diagram

Für weitere Informationen siehe Generelle Probleme und Anmerkungen Punkt 5.

* + IShoppingCartView
  + ICustomerView
  + CheckoutWizardView
  + AddCreditCardWizardView

#### Ergänzte Datenobjekte im Repository

Die hier aufgeführten Datenobjekte sind alle für den Technischen Aspekt von Palladio&Co ergänzt worden.

Ort: cocome-cloud.repository im unteren Bereich bei den DatenTypen

* CreditCardInfo<CompositeDataType>
  + CreditCardNumber<InnerDeclaration>
  + CreditCardType<InnerDeclaration>
* CreditCard<CompositeDataType>
  + CreditCardNumber<InnerDeclaration>
  + CreditCardID<InnerDeclaration>
* List\_creditCard<CollectionDataType>
* CreditCardPin<CompositeDataType>
  + CreditCardPIN <InnerDeclaration>
* StoreID<CompositeDataType>
  + storeID<InnerDeclaration>
* shoppingCart<CompositeDataType>
  + shoppingCartID <InnerDeclaration>
  + itemList <InnerDeclaration>
* customerIdentifier <CompositeDataType>
  + customerID <InnerDeclaration>

#### Ergänzte Datenobjekte im Datamodel

Die hier aufgeführten Datenobjekte sind alle für den Prozessorientierten Aspekt von Palladio&Co ergänzt worden. Dabei wird allerdings dennoch eine Beziehung zu den Technischen Datenobjekten hergestellt.

* CreditCard wurde der Entsprechende Datentyp (aus dem Technischen bereich) zugeordnet
* CreditCardPIN -> DataType: creditCardPIN<CompositeDataType>
* ShoppingCart -> Datatype: shoppingCart<CompositeDataType>
* TransactionID -> DataType: TransactionID<CompositeDataType>
* StoreID -> DataType: StoreID<CompositeDataType>
* Enterprise -> DataType: Enterprise<CompositeDataType>
* EnterpriseCollection -> DataType: List\_enterprises<CompostieDataType>  
   -> Inner Data Objects: Enterprise <Composite Data Object>
* Store -> DataType: Store <CompositeDataType>
* StoreCollection -> DataType: list\_store<CollectionDataType>  
   -> InnerDataObjects: Store <CompositeDataObject>
* CreditCardCollection -> DataType: list\_creditCards<CollectionDataType>  
   -> InnerDataObjects: CreditCard<CompositeDataObject>
* CustomerIdentifier -> DataType: customerIdentifier

#### Änderungen an UseCases:

UseCase 2: ActorStep: customer enters pin ->

Output: CreditCardPIN<CompositeDataObject> (war vorher leer)

ActorStep: cashier pulls credit cardt through card reader ->

Input: creditCard <CompositeDataType>  
Output: -

ActorStep: cashier returns credit card to customer->

Input: -  
Output: CreditCard <CompositeDataType>

UseCase 5: ActorStep „store Manager enters storeID and presses ‘createReport‘ “->

Output: StoreID <CompositeDataObject>

# Generelle Probleme und Anmerkungen

1. Die Runtime-Environment (das innere Eclipse) von Eclipse ist auf meinem Laptop manchmal nicht die schnellste Entwicklungsumgebung gewesen. Das äußerte sich in erster Linie, indem die Usagemodels die ich öffnen wollte mit einem Fehler geöffnet wurden. Allerdings funktionierte das Diagramm tadellos wenn ich es ein zweites mal geöffnet hab. Natürlich vorrausgesetzt, ich habe tatsächlich keinen Fehler gemacht.
2. Falls tatsächlich ein Fehler vorlag, dann wird eine Große graue Fehler-Meldung angezeigt. Da ich manchmal nicht genau hinschau, ist mir leider nie die untere Leiste an diesem Fenster aufgefallen, weswegen ich dachte, ein solcher fehler zerschießt mit das ganze Diagramm. Dem ist jedoch nicht so. Falls ein Fehler vorliegen sollte, kann es über mehrere Reiter in dieser Leiste dennoch geöffnet werden. Beispielsweise über Tree, Tree with Columns und Selection.
3. Die angesprochenen Fehler treten oft auf, wenn ein Model umbenannt wurde. Da es sich nur um umbennen und nicht refactoring handelt, werden die Abhängigkeiten nicht mit verändert.
4. Falls eine Namensänderung zwingend Notwendig wird, können die Resultierenden Probleme mittels Text-Editor behoben werden. Denn die Modelle sind im Hintergrund eigentlich nur XML-Dateien. Wenn also in den fehlerhaften Modellen die Entsprechenden Verbindungen namentlich zur neuen Variante geändert werden, dann funktionieren die Abhängigkeiten wieder. Natürlich kann der Aufwand dabei extrem schwanken. Wenn das Repository umbenannt werden würde, würden ALLE darauf zugreifenden Modelle fehlerhaft werden, weil sie das Repository nicht mehr finden. Somit müssten auch ALLE Modelle per Text-Editor bearbeitet werden…
5. Wenn das composite-Diagramm neu generiert wird, dann wird es in einem sehr kleinen Fenster dargestellt. Aufgrund dessen gehen allerdings Informationen verloren. Um dies zu vermeiden, einfach das Fenster groß ziehen und dann einen rechtsklick in das Diagramm und dort „ArrangeAll“. Dann werden die Elemente ein wenig auseinandergezogen. Sollte die auswahlmöglichkeit nicht vorhanden sein, dann muss darauf geachtet werden, dass der Innere Rahmen des Diagramms angewählt ist. Das kann man nur im oberen Bereich des Diagramm feststellen.

# Verwendete Ressourcen

|  |  |  |
| --- | --- | --- |
| Ressource | Version | Beschreibung/ Nutzen |
| Eclipse | Oxygen 3a bzw. 4.7.3a | Stark erweiterbare IDE |
| Palladio-Plugin | Vom 28. Mai 2018 | Plugin für Eclipse zum Verwenden von Palladio |
| Bppcm.ecore | Vom 05. Julie 2017 | Meta-Modelle von IntBIIS |
| Pcm.ecore | Revision 34847 | Meta-Modelle von Palladio |