Applying SOMA Method to IBM's Industry Models

Brian Byrne (IFW/IAA Process and Integration Models Architect)
Shuvanker Ghosh (SOMA Core Team and Sr. IT Architect, Financial Services)
Ali Arsanjani (IBM Distinguished Engineer, Chief Architect, SOA & Web Services Center of Excellence)

Release 2.0.1

V2.0.1 Edition (October 2007)
This edition applies to v2.0 and to all subsequent releases and modifications until otherwise indicated in new editions or Technical Newsletters. Changes are made periodically to the information herein; any such changes will be reported in subsequent revisions or Technical Newsletters.

Preface

About this Book

This publication provides methodological guidance relating to the use of IBM's Industry Models as part of a strategic approach to defining a Service Oriented Architecture using the SOMA method. This focuses on the SOMA 3.1 method – with an emphasis on the phases of *Identification*, *Specification, Realization and Implementation.*

This publication makes detailed references to the models that make up the IBM Industry Models and knowledge of the nature and content of these models is assumed. Similarly, while this publication provides a high level overview of the SOMA 3.1 method, it is not the core source of guidance on the full scope of this method. The goal of this document is to provide Industry Model specific guidance on top of the SOMA method in order to illustrate how SOMA can be used to enable Industry Model based projects.

The primary additions in this edition of previous Industry Models / SOMA discussions is support for informational modeling and information as a service, and the addition of rules analysis and design.

Who Should Read This Book

This book provides essential guidance for analysts and designers working with the Industry Models in the context of a SOMA engagement. This will include the following roles and skills within the organization:

Strategists and Planners

Those involved in the initial inception and evaluation of projects, and those concerned with potential overlap and producer/consumer relationships between projects. The initial chapters of this book will provide the greatest guidance to this audience, detailing the use of the foundation models to capture and analyze project scope and project dependencies.

Business Modelers

Those involved in defining models at a sufficiently detailed level to clearly capture business requirements. The chapters of this book detailing the identification and customization of processes, and on to the modeling of use cases within the Business Object Model (BOM) will provide guidance to this audience.

Business Analysts

Those involved in the detailed analysis of subsystems at a logical level will be intimately involved in the definition of use cases and the supporting types within the Business Object Model (BOM), completing the definition of business requirements for the full scope of the project in hand.

Solution Architects

Those responsible for the overall solutions architecture of a project will be consuming the requirements as expressed within the Process models and BOM and producing a solution architecture that has a service based pattern derived from the constructs of the Interface Design Model (IDM).

Solution Designers

Solutions designers consuming the detailed requirements as expressed in the Interface Design Model will define detailed service definitions as part of component architecture.

Related Documentation

1. Industry Model User Guides
2. SOMA 3.1 Information discipline technique paper available at https://w3.webahead.ibm.com/w3ki2/display/ISAS/SOMA+Information+Discipline
3. SOMA 3.1 Information discipline tool mentor paper available at https://w3.webahead.ibm.com/w3ki2/display/ISAS/SOMA+Information+Discipline
4. SOMA Method available for download from MethodWeb at: http://method.ibm.com
5. SOMA 3.1 Core Tasks: Practitioner’s Guide
6. Overview and Usage of Solution Templates and Patterns in SOMA

Contents

Figures

Introduction

1.1. What are IBMs Industry Models?

IBM's Industry Models are a collection of interrelated models addressing different aspects of the analysis and design of applications or integration solutions for key industries such as Banking (IFW) and Insurance (IAA). Of particular interest within this guide are those models that relate to the analysis and design of a Service Oriented Architecture to support the operational needs of an organization.

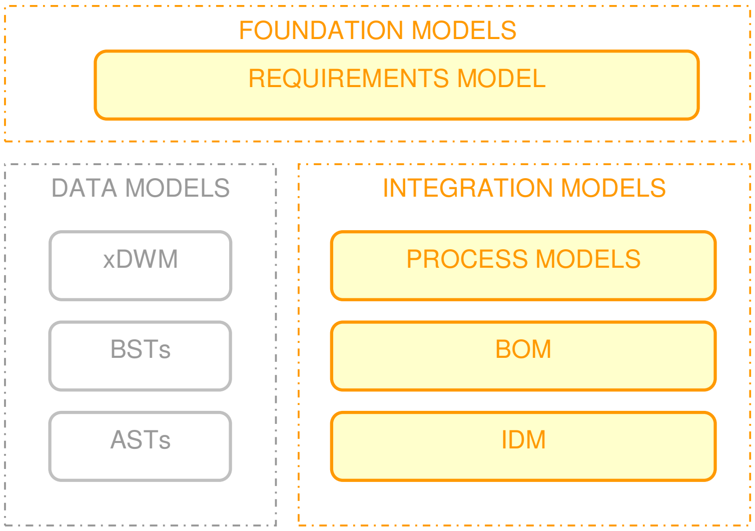


Figure 1. The Industry Models relating to service oriented architectures

The Industry Models are structured as a set of foundation models that act as a standard taxonomy upon which the other offerings are built. Derived from these foundation models are detailed process analysis models detailing the end to end processes required to support an organizations business. Derived from these in turn are detailed service analysis (BOM) and service design (IDM) models that act as a blueprint for SOA and component based projects within an organization.

It is important to understand that the Industry Models do not provide an "out of the box" solution. The Industry models are a deliberate abstraction from the specifics of any one organization, in any one problem domain. The models provide the blueprints upon which the specific processes and underlying services that support the business may be constructed.

Similarly, a customized set of models that fully capture the requirements and resulting designs of the modeled organization do not provide a full runtime solution for the project. The goal of the Industry Models and of enterprise modeling in general, is to provide consistent patterns and artifacts across a wide range of the enterprise. This approach to modeling greatly improves the chances of producing a truly reusable service based infrastructure. It does not however bypass the implementation stages of software development. The patterns described within IDM provide invaluable guidance to those seeking to expose, for example, the capabilities of existing CICS transactions as reusable XML messages, but the task of writing the COBOL code to translate from an XML message into transaction invocations still exists.

Each individual financial institution, and indeed, each individual project within that institution will have a distinct set of challenges relating to the target operating environment. These challenges range from the limitations of existing systems and infrastructure, through to the challenges of the selected solution architecture. The challenges encountered in deploying enterprise wide services will differ from the challenges associated with the construction of an enterprise wide J2EE architecture. Each of these sets of challenges can introduce compromises to the final design to account for limitations in the operational environment or other concerns. These compromises are captured within the design model of IDM, leaving the BOM intact. This separation of analysis from design is critical to maintain a clear picture of business requirements, and their relationship to constantly evolving designs.

The Industry Models support the identification, analysis and design of requirements' based solutions, irrespective of the target environment. The technology-based challenges of enterprise integration vary with the selected infrastructure. The challenges of capturing and expressing business requirements and of deriving service architecture from these requirements do not. It is the intent of the models to support the requirements based analysis and design of systems in a way that is not biased by particular solution architecture. However, capabilities are provided to transform and deploy model artifacts into technology specific domains, for example a BPEL / J2EE environment.

1.2. What is SOMA?

SOMA is an end-to end SOA development method aimed at enabling business processes through the identification, specification, realization, implementation and deployment of business-aligned services, service components, and flows that form the SOA foundation. The SOMA method defines prescriptive tasks and provides in-depth guidance on how to move from the business models created through business and strategy analysis, such as the output from the IBM Component Business Modeling (CBM), to the IT models required by an SOA. Application of the SOMA method to identify, specify and implement services can be accelerated by leveraging IBM's Industry Models. IBM's Industry Models can be extended and customized in line with organizations specific requirements during application of SOMA method to identify, specify, and implement services.

SOMA drives continuity between business drivers and IT implementation by extending business characteristics (e.g. goals and key performance indicators) into the IT analysis and architectural decisions. The modeling, analysis and design performed during SOMA, leveraging the Industry Models, is platform agnostic, but establishes a context for making platform specific decisions in later phases of the lifecycle.

The SOMA method consists of five distinct phases – Identification, Specification and Realization, Implementation, Deployment. Each phase of SOMA consists in turn of detailed steps with associated methodological guidance. The Industry Models accelerate the top down identification of services. The detail analysis model (BOM) within the Industry Models accelerates specification of services. The design model (IDM) and the derivative Java Design Model (JDM) within the Industry Models accelerates specification and implementation of services.

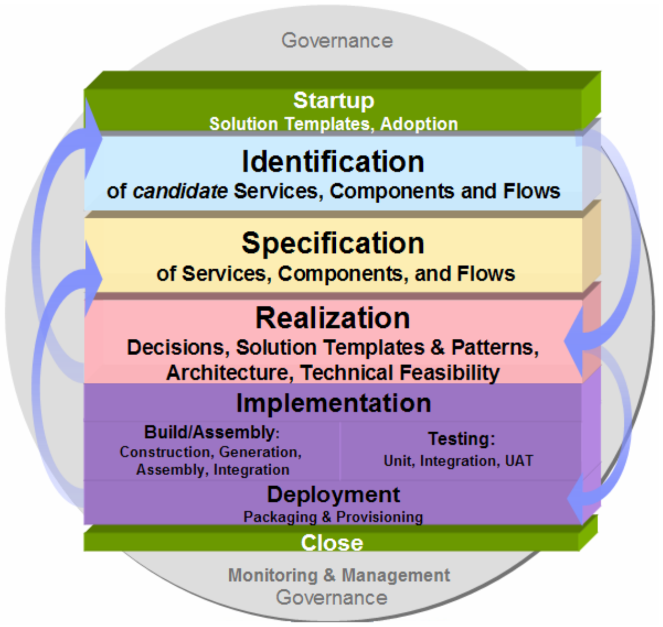

Governance
Startup
Solution Templates, Adoption
Identification
of *candidate* Services, Components and Flows
Specification
of Services, Components, and Flows
Realization
Decisions, Solution Templates & Patterns, Architecture, Technical Feasibility
Implementation
Build/Assembly:
Construction, Generation, Assembly, Integration
Testing:
Unit, Integration, UAT
Deployment
Packaging & Provisioning
Close
Monitoring & Management
Governance

Figure 2. SOMA phases

The application of the SOMA method to Industry Models such as IAA and IFW is described here with a focus on the Identification and Specification and Implementation of services. Each of these phases consists of distinct activities which drive aspects of the analysis/design lifecycle. It is important to realize that the execution of these phases and the activities within these phases is iterative. The sequencing provided within this document may be altered within the scope of a given project.

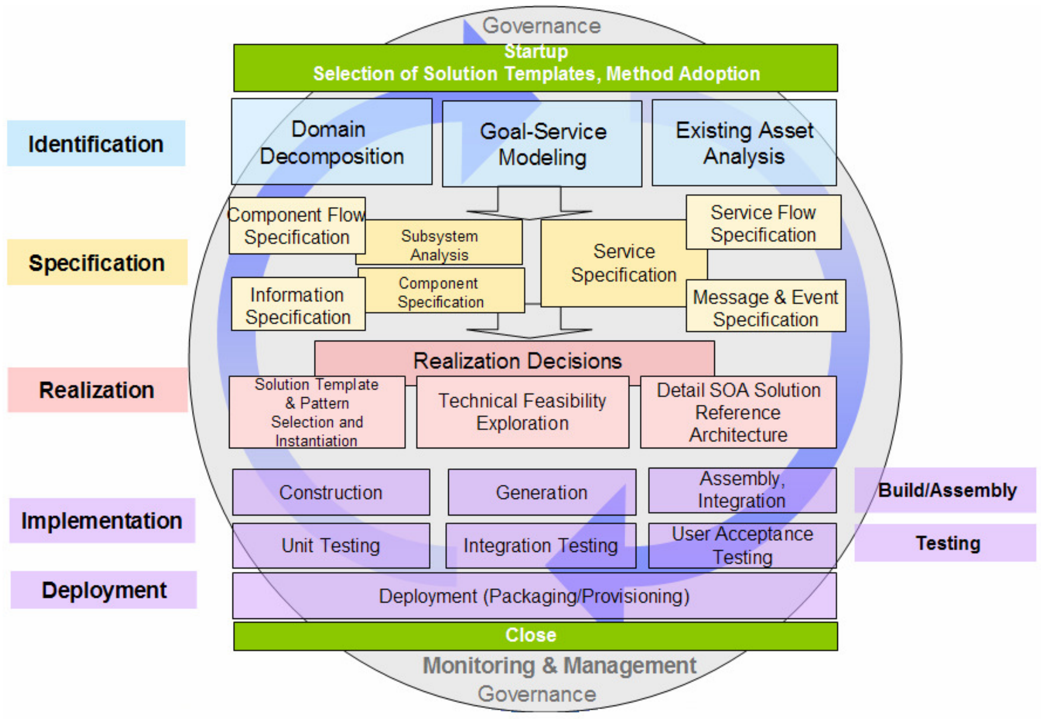


Figure 3. Detailed SOMA activities

1.3. Scope of this Document

The scope of this document is to highlight the specific tasks within SOMA that are relevant to the usage of the Industry Models, and the specific considerations beyond the core assertions of SOMA that are relevant to an Industry Models specific engagement.

It is not the goal of this document to provide a detailed description of SOMA, or of the Industry Models, the goal is to describe the detailed analysis and design tasks that must be considered to fully leverage the Industry Models.

Chapter 2. Solution Startup

Solution startup focuses on selecting solution templates and patterns that guide the overall SOMA initiative and conducting method adoption workshops in addition to business-as-usual tasks for project management and project initiation.

Every SOA engagement tends to have unique characteristics and each SOA solution is a hybrid of various solution types. The SOMA 3.1 introduces the notion of solution templates to define additional tasks, roles, and work products in the method necessary to support recurring solution types. A solution template is a mechanism for extending SOMA to incorporate and combine multiple strategies and approaches for building an SOA solution, based on a particular set of constraints or patterns, such as integrating systems, or package applications. Services that are identified and specified by applying SOMA can be realized by Composite Business Services (CBS), custom development, legacy integration and transformation, or package application integration.

Every SOMA Phases has insertion points or slots for extending the SOMA Delivery Processes by adopting solution templates and for inclusion of activities and tasks from the solution templates. These slots in the SOMA Phases and Delivery Processes provide variability in the method and allow specific activities and tasks to be inserted in the method flow for specific solution in an engagement.

In SOMA 3.1 following solution templates are defined:
1. Solution template for CBS asset construction
2. Solution template for CBS asset consumption
3. Solution template for custom service component development

In subsequent SOMA releases, additional solution templates will be defined:
1. Solution template for legacy integration and transformation
2. Solution template for package application integration
3. Solution template for SOA governance
4. Solution template for service oriented integration
5. Solution template for performance engineering

Note that in this context, “Solution Templates” refers not to the reporting views in front of the Industry Model warehouse offering (e.g. BSTs).

2.1. Selection of Solution Templates and Method Adoption

Each of the solution templates that are provided by SOMA out-of-box is evaluated and a subset of the available solution templates is selected during solution startup. This selection tends to be guided by project specific concerns, based on information about the solution available during solution startup.

Using the selected solution templates and client methods and processes as input, a Method Adoption Workshop (MAW) is executed to tailor the SOMA method to fit the needs of the client and the engagement.

During Method Adoption Workshop, the activities from each of the selected solution templates are injected into SOMA Phases in an SOMA Delivery Process to create an integrated end-to-end delivery process for the specific engagement.

Chapter 3. Identification

The Identification phase of SOMA involves an analysis of the business domain in question, existing capabilities and assets and variation in business information and logic with a view to identifying distinct types of service candidates:

- Process Services – services that encode meaningful business logic
- Information Services – services that provide access to information
- Rule (Decision) Services – services that encapsulate decisions or rules
- Infrastructure Services – services that support platform level support such as scheduling and logging[1]

This section of this paper describes the Identification phase of SOMA with respect to the Industry Models.

3.1. Initiate SOMA Identification

Each SOMA phase includes a set of preparatory tasks, which incorporate the setup for that phase. SOMA Identification is initiated by performing preparatory tasks such as confirming the results of the method adoption workshop, analyzing risks for the phase, refining project estimates based on up-to-date facts and information available about the project, and establishing guidelines for this phase on the initiative.

3.2. Instantiate Solution Templates and Patterns

At this stage we are ready to instantiate solution templates and patterns for this phase. This activity defines insertion points/slots for extending the SOMA Identification phase to incorporate multiple strategies and approaches using as defined by solution templates. These placeholders were replaced by defined activities and tasks from corresponding solution templates during the method adoption workshop as described in Chapter 2 - Solution Startup.

3.3. Goal-Service Modeling

The primary objective of Goal-Service Modeling is to align services with business goals and to facilitate identification and prioritization of services. In addition, Goal-Service Modeling is used to filter out those services that do not meet business goals.
Goal-Service Modeling associates services with business goals and identifies services by analyzing the goals of an enterprise or a business unit. This technique starts with generalized statements of business goals, with subsequent decomposition into sub-goals that must be met in order to fulfill the higher level goal. This eventually leads to a level of decomposition that supports the identification of services associated with the sub-goals. Often goal service modeling may be performed outside of the scope of any specific project, and results are used to define an overall services landscape, which in turn can be used to drive the identification of projects and their dependencies.
During this process, KPIs are identified and collected; along with the metrics that will be used to measure, monitor and verify the degree of success in actually achieving the

[1] Infrastructure services are considered to be largely platform and environment specific and are not discussed as part of this paper.

goals by building and deploying these services. Goal-Service Modeling establishes the conceptual link between services and business goals, as well as the means to demonstrate whether the goals are actually being achieved once the services are implemented. It also narrows the focus of analysis for later steps of SOMA.

Using the stated strategic plans of the organization, or any available documents that describe what the organization seeks to achieve from this project, review these goals of the project. These stated goals and objectives should be distilled into distinct strategic statements that will provide guidance for the project moving forward. Strategic statements, when viewed across multiple projects, should not conflict or contradict each other. These strategic statements should also be used to derive any process design criteria (e.g.: non functional requirements) which can be used to verify the quality of future process designs.

Goal Service Modeling ensures that the application of SOMA remains focused on the key goals and objectives and the business components or business processes that will help attainment of the business goals. IBM's Component Business Model (CBM)[2] can play a key role in the identification and documentation of business and IT strategy. CBM allows business strategy to be expressed in terms of key functional elements known as business components, and results in defined goals and objectives that can be used to guide process scoping and customization.

Goal-Service Modeling helps with the discovery of business aligned services, through the clear articulation and decomposition of business goals, and the mapping of those goals to processes and services within the Industry Models. This approach also provides an effective mechanism for narrowing the focus of the service identification initiatives, providing a workable scope for projects.

Goal-Service Modeling starts with stated business goals, typically the output of a CBM engagement, which can then be decomposed to derive key performance indicators (KPIs), allowing the performance of identified processes and services to be measured in a meaningful way against business objectives.

3.3.1. SWOT Analysis

SWOT (strengths, weaknesses, opportunities, threats) analysis is a method that supports the review of the business processes of an organization. SWOT analysis is used in the early stages of the strategy and project prioritization, and helps direct focus on key issues that are facing that organization.

During SWOT analysis, once key issues are identified, they are cross-matched with the stated strategic objectives of the organization. The goal of this exercise is to identify the strengths of the organization that can be built upon, and the weaknesses that need to be corrected.

[2] CBM takes a 3-phased approach to develop and implement an IT Strategy. The *Insight Phase* emphasizes the development of the end-state business component model. The *Architecture Phase* determines the gaps between the current state and the end state from a process, systems and organization perspective. The *Investment Phase* develops the roadmap to the target environment and the business case for the priority initiatives.

A strength could be: specialist expertise, new or innovative products or the quality of processes and procedures.

A weakness could be a high latency in processing customer enquiries, a lack of available information relating to existing systems or a lack of required skills.

Opportunities and threats are external factors, such as a developing market or a competitor having superior access to distribution channels.

SWOT analysis can be very subjective, and should not be relied upon too heavily, but it can provide a valuable insight into the factors affecting the success of a project.

The following are some simple rules to guide SWOT analysis:

- be realistic about strengths and weaknesses
- distinguish between where the organization is today, and where it could be in the future
- avoid grey areas and focus on specifics
- keep it short and simple. Avoid complexity and over analysis
- support conclusions with quantifiable historical data

SWOT Analysis is a valuable mechanism about which requirements discussions can be structured, providing some of the essential detail required to customize process models to the future needs of the organization.

3.3.2. Root Cause Analysis

Root Cause Analysis refers to the identification and solving of the true cause of a problem, rather than being reactive to symptoms of that problem.

Consider the example of a customer contacting the organization because a check book has not been shipped. The common reaction is to track down the check book order in question, and assign it a high priority to expedite provision of the check book. What is usually not addressed is that in expediting the order, other customers orders are being delayed because the process was disrupted to get this order processed, which does little more than to perpetuate the cycle. This results in an “Addiction” cycle, where the organization as a whole becomes addicted to intervening in order processing in order to satisfy customer needs, rather than addressing the underlying cause of the delays.

A more appropriate response to this challenge is to identify why the first order was delayed in the first place, and to determine whether this was a rare event, or a problematic trend. However, this is seldom the chosen response, as the challenge facing the operator in question was simply to expedite one customer order.

Undesirable situations of this nature within organizations are usually 95% related to process problems and only 5% related to personnel problems. However organizations seldom achieve the benefit they could gain from understanding the root cause of the problem they are facing.

Using the Industry Models as a framework within which the organizations' problems can be discussed is an excellent way to identify the Goals, KPIs and Root Cause underpinning specific issues, in a neutral, business focused environment.

3.3.3. KPI Analysis

At this stage of Goal Service Modeling analysis is performed to identify the metrics by which the business process may be assessed. The nature of these metrics is dependent on the process involved, but they should relate strongly to the goals of the business that the process addresses. For example, if the objective of a business process is to provide an offer to a customer, then the key performance indicators (KPIs) for that process would be the duration that provision of that offer takes, the cost of doing so, the percentage of failures to provide an offer and so on. Identification of KPIs follows a process of evaluating the objectives of all or part of the processes and identifying the facts that indicate the success of that process in the light of these objectives. Identifying these KPIs is critical for closed loop analysis of business processes, and the ongoing refinement of business processes to better meet business objectives.

3.4. Domain Decomposition

Domain decomposition is one of the complementary techniques in SOMA to identify candidate services. It is a top down approach for identifying services and consists of the following key activities: Functional Area Analysis, Process Decomposition, Information Analysis and Modeling, Variation Oriented Analysis, and Rule and Policy Analysis. Domain Decomposition activities in SOMA are accelerated by leveraging Component Business Model, Business Process Model, and Business Object Model from the Industry Model as illustrated in the figure below. Note that the execution of these discreet steps may be iterative in nature.

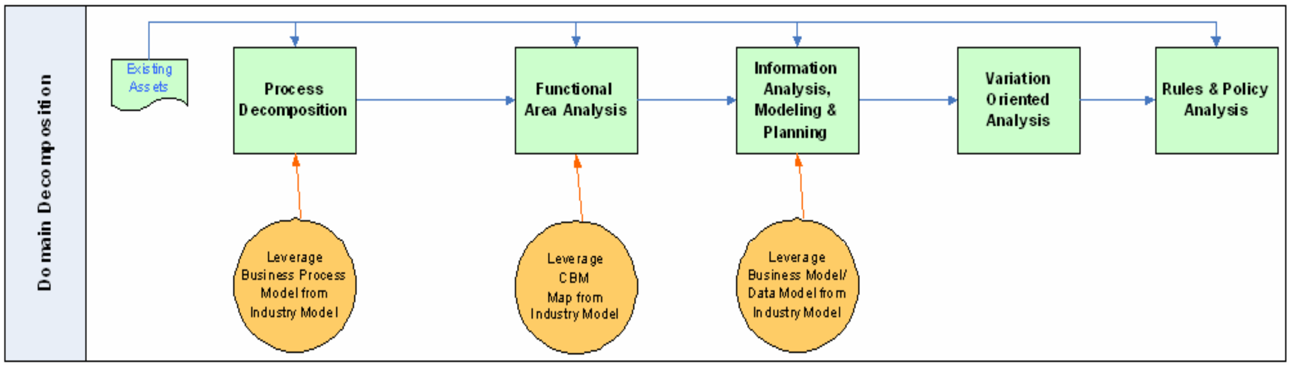

Domain Decomposition
Existing Assets
Process Decomposition
Functional Area Analysis
Information Analysis, Modeling & Planning
Variation Oriented Analysis
Rules & Policy Analysis
Leverage Business Process Model from Industry Model
Leverage CBM Map from Industry Model
Leverage Business Model/ Data Model from Industry Model

Figure 4. Model elements leveraged during domain decomposition

3.4.1. Functional Area Analysis

At this early phase in the project lifecycle, it is essential to clearly define project scope, and to identify how this project scope relates to the scope of other projects within the enterprise.

It is highly recommended, before a project starts, that a functional and an informational scoping be carried out for that project, allowing multiple projects to be analyzed through common business definitions, to determine any dependencies, inconsistencies or duplications of effort amongst those projects.

Functional Area Analysis relies on a partitioning of the business such as defined by the IBM Component Business Modeling technique (CBM) as an initial reference against which business domains may be identified and discussed. Further decomposition of these business domains will yield individual business functions and then business tasks that are required to support each function.

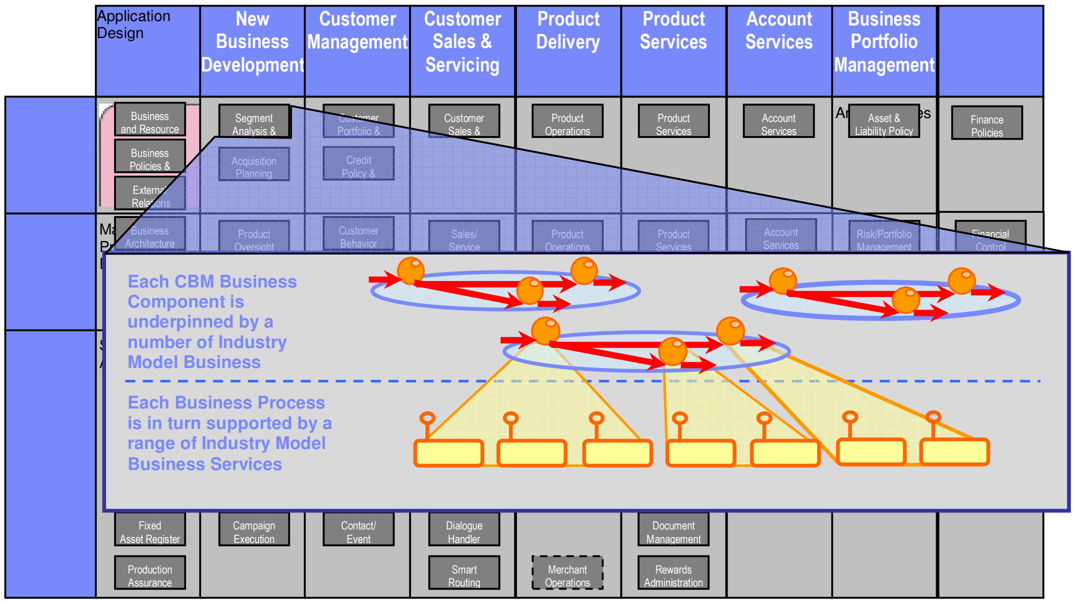


Figure 5. Functional area analysis using CBM as input

In addition to scoping and prioritization, functional area analysis defines business boundaries for IT subsystems and their corresponding service components that realizes the services. Both SOMA and the Industry Models use functional areas to categorize candidate services, making service catalogs more manageable. Business functions are also used within the Industry Models to suggest potential subsystem boundaries within a component model. During functional area analysis, key entities supporting the functional areas of interest are also identified in addition to the business functions themselves.

Functional scoping provides a valuable input to the scoping of business processes. For each functional area that has been scoped as “core” to the project in question, a set of processes and activities within the Industry Process Models can be identified that support that function. If managed incorrectly, the relationship between function and process/activity can be many to many, thus there are many functional areas that will result in a given process being scoped, and conversely, many processes may support one function. However, identification of the correct granularity of business function for the elements being categorized yields valuable results. Functional analysis can also assist in the identification of the ultimate ownership of defined services within a large organization.

Scoping of processes involves testing, through discussion, the relevance of each process to the project in hand. This can be done through consideration of:

- the organization unit performing the work
- present methods of the process versus the “to-be” state
- the roles involved in the process execution
- the product types being supported by processes
- the customer segment being served by the processes

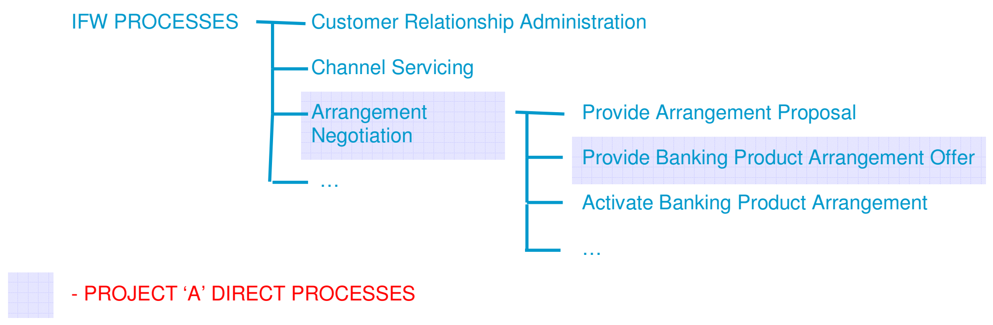

IFW PROCESSES
Customer Relationship Administration
Channel Servicing
Arrangement Negotiation
…
Provide Arrangement Proposal
Provide Banking Product Arrangement Offer
Activate Banking Product Arrangement
…
- PROJECT 'A' DIRECT PROCESSES

Figure 6. Example process scoping

Functional Area Analysis provides a short list of areas of responsibility that will require some new capability or support from the project solution. This scoped list forms an extremely useful completeness test, i.e. for every function scoped; there should be at least one activity in a workflow to support it.

Additionally, processes can be prioritized for the project according to those that support functions most closely aligned with stated business goals and objectives. Processes that support functions, that are deemed to be peripheral to the project, are only of relevance if this project is to act as a requirements input for the delivery of these peripheral functions.

3.4.2. Process Decomposition

Having identified the set of processes that are required in the scope of the project, the business analysis team will customize each process to meet the specific needs of the organization. The Industry Process Models can be used to seed process analysis within SOMA, in order to define these processes that will support a project. These customizations involve analysis of a number of aspects of the process as a whole, the context of the process, and of the constituent activities of that process.

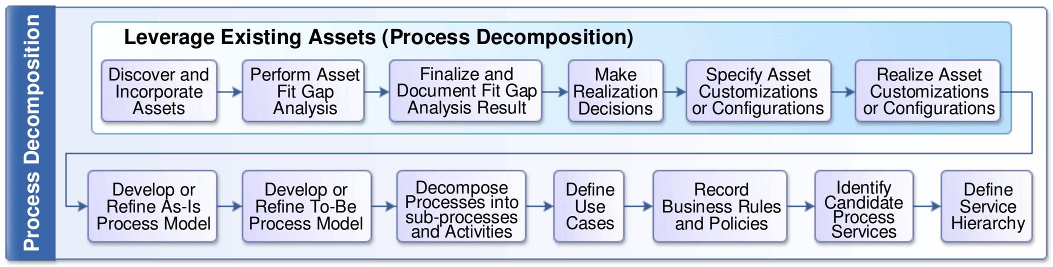

Process Decomposition
Leverage Existing Assets (Process Decomposition)
Discover and Incorporate Assets
Perform Asset Fit Gap Analysis
Finalize and Document Fit Gap Analysis Result
Make Realization Decisions
Specify Asset Customizations or Configurations
Realize Asset Customizations or Configurations
Develop or Refine As-Is Process Model
Develop or Refine To-Be Process Model
Decompose Processes into sub-processes and Activities
Define Use Cases
Record Business Rules and Policies
Identify Candidate Process Services
Define Service Hierarchy

Figure 7 SOMA defines steps to leverage Industry Models during process decomposition and other activities

The following key steps are performed to leverage industry models during process analysis:

- Perform detailed process analysis
- Perform fit gap analysis as relates to clients as-is business processes

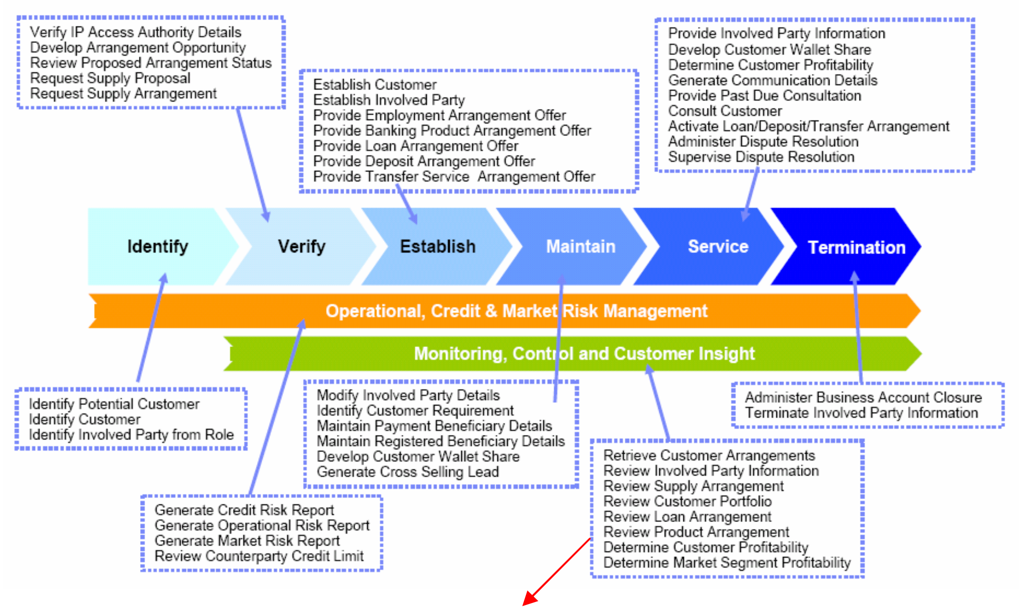
- Customize business processes to fill gaps
- Develop to-be process model
- Select and extend use cases in BOM directly derived from activities within the process models

Use of the Industry models also enable detailed process analysis resulting in end-to-end business workflows across the scope of an organization and provide a fast start to capturing the detailed processes that will support future business strategies.

Process decomposition involves the analysis of business process and sub process candidates to identify the nesting and dependency between these processes, and is critical to ensure that proposed services are aligned to consumer needs rather than the needs of any service provider, and are expressed at an appropriate level of granularity.

Note that the Industry Process Models are not intended to be "ready to run" workflows. They are intended to provide generic, leading practice templates which an organization will customize by adding the specific requirements that will underpin the organizations unique competitive advantage.

Analysis of each business function identified through Functional Area Analysis will yield a set of processes which are required to support each function. Note that many of these processes will in fact cross functional boundaries and as such will be identified through the decomposition of more than one function. In effect, it is the constituent activities of these business processes that can be assigned to business functions (such as CBM components).


Verify IP Access Authority Details
Develop Arrangement Opportunity
Review Proposed Arrangement Status
Request Supply Proposal
Request Supply Arrangement
Establish Customer
Establish Involved Party
Provide Employment Arrangement Offer
Provide Banking Product Arrangement Offer
Provide Loan Arrangement Offer
Provide Deposit Arrangement Offer
Provide Transfer Service Arrangement Offer
Provide Involved Party Information
Develop Customer Wallet Share
Determine Customer Profitability
Generate Communication Details
Provide Past Due Consultation
Consult Customer
Activate Loan/Deposit/Transfer Arrangement
Administer Dispute Resolution
Supervise Dispute Resolution
Identify
Verify
Establish
Maintain
Service
Termination
Operational, Credit & Market Risk Management
Monitoring, Control and Customer Insight
Identify Potential Customer
Identify Customer
Identify Involved Party from Role
Modify Involved Party Details
Identify Customer Requirement
Maintain Payment Beneficiary Details
Maintain Registered Beneficiary Details
Develop Customer Wallet Share
Generate Cross Selling Lead
Administer Business Account Closure
Terminate Involved Party Information
Generate Credit Risk Report
Generate Operational Risk Report
Generate Market Risk Report
Review Counterparty Credit Limit
Retrieve Customer Arrangements
Review Involved Party Information
Review Supply Arrangement
Review Customer Portfolio
Review Loan Arrangement
Review Product Arrangement
Determine Customer Profitability
Determine Market Segment Profitability

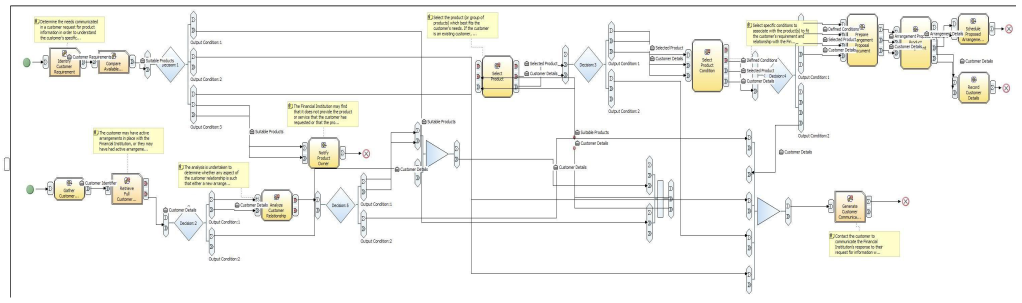


Figure 8. Process decomposition example

Business process reengineering initiatives are often the impetus for customizing workflows, in order to improve the customer experience, make the processes more effective, and to reduce process defects, as in Six Sigma projects. Such initiatives often lead to workflow automation and new requirements for access to information via service-based architectures.

Having identified the set of processes that are required in the scope of the project, the business analysis team will customize each process to meet the specific needs of the organization. These customizations involve analysis of a number of aspects of the process as a whole, the context of the process, and of the constituent activities of that process.

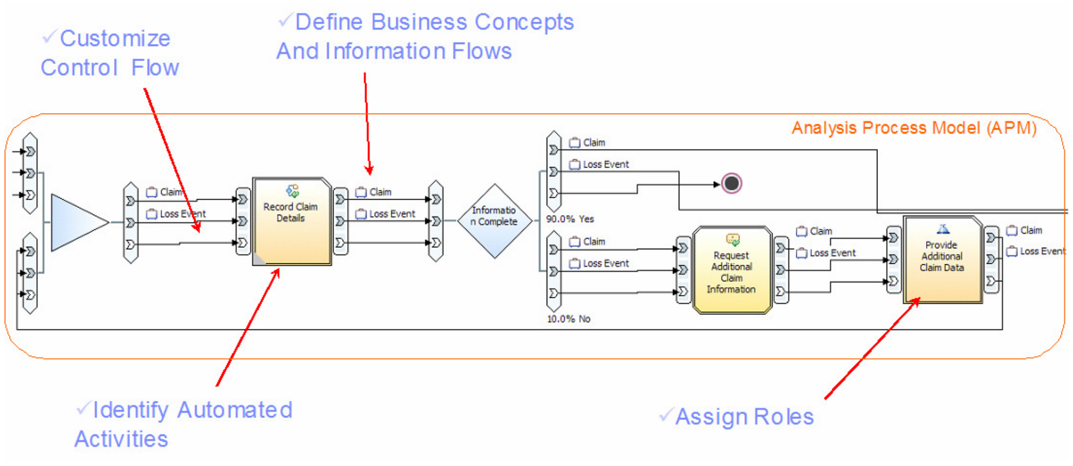

✓Customize Control Flow
✓Define Business Concepts And Information Flows
Analysis Process Model (APM)
Claim
Loss Event
Record Claim Details
Information Complete
90.0% Yes
10.0% No
Request Additional Claim Information
Provide Additional Claim Data
✓Identify Automated Activities
✓Assign Roles

Figure 9. Customization of business processes

The following categories of changes are performed to customize the business process to fit the organizations needs:

- Customize Control Flow
- Define Business Concepts And Information Flows
- Identify Automated Activities
- Assign Roles

Define Business Use Cases

An excellent way of ensuring that all requirements have been captured for a process or activity, and to attach extra information to the modeled elements, is to complete a business use case for the activity or process (and constituent activities). This is a structured document that allows essential information relating to an activity to be captured in a formatted way

An example of business use case template is provided below:

Activity Name	A unique name that specifically defines this activity, process or nested process, using a standard verb operating on a business object
Summary Activity Definition	A concise but accurate business definition, usually of the form "The business purpose of this Activity is to…"
Detailed Activity Description	A very detailed and unambiguous description of what happens within the activity. This definition should make it clear what is in scope and what is out of scope for this activity, and what is achieved for the business by this activity. The definition may take the form: "This Activity (String) encompasses the activities necessary for <Organization Name> to…" and may include a bulleted list of the specific responsibilities of this activity, or the specific context describing why this activity is necessary to the business, to achieve certain goals, objectives and functions relevant to this process.
Input Informational Requirements	A business definition of the information that is required as input for this activity to function, or what information must be provided before this activity can begin.
Output Information Requirements	A business definition of the information that is produced as an output of this activity, or what has changes for the business because this activity has completed.
Business Rules	A definition of any specific rules pertaining to this activity. These are often provided as a bulleted list, or "if…then" conditional statements that describe how the activity is to be executed. These may alternatively be a reference to a specific business rule that is described separately.
Critical Success Factors	A definition of any preconditions or success factors that must exist for this activity to execute successfully. These are often provided as a bulleted list. This may include a description of innovated areas that will be done differently in the new process.
Constraints	A definition of any known limitations or restrictions on the

	process. This may include legal or regulatory requirements as they pertain to this activity.
Dependencies	A definition of any logical units or other processes upon which this activity will depend for its operations. This includes policy changes and organizational decisions that must be made in order to implement this activity.
Non-Functional Requirements	Definitions of any supporting requirements that are not directly involved in the business definition of this activity, but still affect the successful outcome of this activity, such as key performance indicators and systems support requirements for the business.

Figure 10. Business use case template

These business use case documents capture additional information about the constituent parts of the customized process that are not evident from the process itself, and provide essential input information to the definition of use cases later in the customization of the Industry Models.

An example of a business use case for the activity "Compare Available PD Dtls with Cust Rqmts" is shown below.

Activity Name	Compare Available Product Details with Customer Requirements
Summary Activity Definition	The business purpose of this Activity is to examine product details with reference to customer requirement details in order to identify resemblances or differences.
Detailed Activity Description	This activity encompasses the activities necessary for <Organization Name> to compare the details of the customer's requirement with the products or product groupings offered by the Financial Institution which seem likely to meet this requirement. Try to find the best possible match with the customer's needs from amongst these products.
Input Informational Requirements	One or many customer requirements. These requirements are selected from a list of predefined customer requirements that <Organization Name> specifically tracks and cross references to product offerings
Output Information Requirements	One or many product descriptions that meet the needs of one or all of the supplied customer requirements. Only products that are actually available for sale should be returned.
Business	• The supplied customer requirements should be used to

Rules	match against the products of the organization • Only products that are in an "Available" state should be returned (Note that these may alternatively be references to specific rule requirements stated elsewhere)
Critical Success Factors	• Product development and review must include the cross referencing of a list of possible customer requirements with the product being developed, to that this can be later used as a lookup between customer requirements and products. • The list of possible customer requirements must be maintained over time • Customer requirements and product details must be cross-referenced in such a way as to product useful results from a lookup. No benefit is achieved if all lookups retrieve all, or nearly all, available products.
Constraints	• Customer requirements can only be selected from a pre-selected list, as only items in this list will be cross-referenced to product data.
Dependencies	• Product system must be capable of storing the customer requirements satisfied by a given product • These requirements must be set during product development and review • A system must be available to maintain possible customer requirements
Non-Functional Requirements	No stated non-functional requirements.

Figure 11. Example activity description document

Initiating Scenarios

Analysis of the initiating scenarios of a process is critical to understanding the circumstances that lead to the initiation of a particular process or activity. Under what circumstances will the process start and what will be the logical relationship between these circumstances? Analysis of the initiating scenarios should yield an understanding of the specific triggers that cause the process to start, and of the relationship between these triggers, expressed as a logical combination of events. Note that the form of this logical expression varies from tool to tool, but is typically expressed through the use of branching logic and decisions within a process modeling tool.

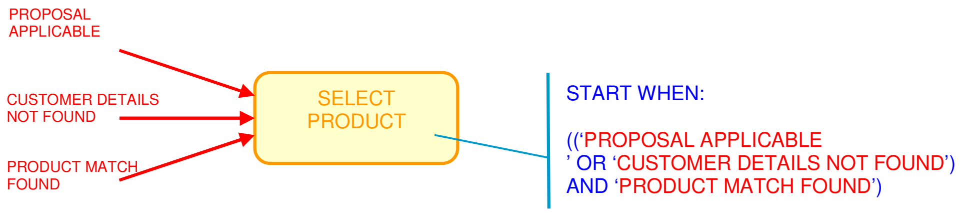

PROPOSAL APPLICABLE
CUSTOMER DETAILS NOT FOUND
PRODUCT MATCH FOUND
SELECT PRODUCT
START WHEN:
((‘PROPOSAL APPLICABLE ’ OR ‘CUSTOMER DETAILS NOT FOUND’) AND ‘PRODUCT MATCH FOUND’)

Figure 12. Example initiating scenario

Initiating scenarios should be analyzed for the launch of a process, as well as for each activity or nested process within the processes.

Completing Scenarios

Analysis of completing scenarios is essential to understanding the termination conditions of a process or activity. What are potential results of execution, and what is the logical relationship between these results? Analysis of the completing scenarios of a process should yield an understanding of the specific triggers that are initiated when a process ends, and of the logical relationship between these triggers, expressed as a logical combination of events. Again, the form of these will be tool dependent – but is typically expressed through the use of flow control logic.

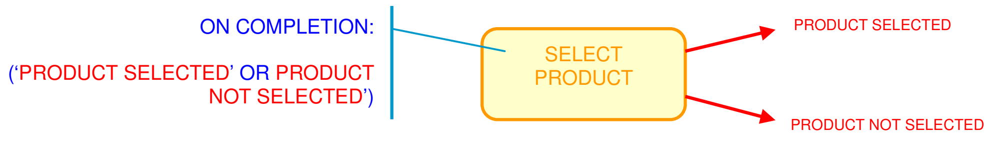


Figure 13. Example completing scenario

Completing scenarios should be analyzed for all activities and nested processes within a process, as well as for the end of the process as a whole.

Activity Customization

At this point in model customization, each activity within the scoped process should be challenged for its applicability and validity to the organization and specifically the project under analysis. This will result in the addition of new activities and the removal of unnecessary existing activities from the critical business process. Where a new activity is required within the flow, existing pre-defined activities within the process model repository should be reviewed for applicability, and reused where possible.

The granularity of naming and definition is an important concern here. The temptation will usually be to define and add an activity that meets the very specific needs of the current analysis pattern. In many cases, however, a slightly broader, more generic activity definition would result in a more reusable construct. Care must also be taken not to construct too broad a definition that is too far removed from the actual business requirements. Activity definitions are a careful balance between the opposing factors, high applicability to business context, and high reuse potential. For this reason, when defining a new activity, it is useful to consider what other contexts this activity may apply to, and to try to define activities with an enterprise wide view in mind.

When adding new activities it is important to retain an analysis perspective on the process being customized. Try not to jump to a solution, or to define the in terms of underlying systems. The goal of this level of business analysis is to define the business problem independently of any specific solution to that problem. Considering the process as it exists today, or as it is manifested within a particular system, will unnecessarily constrain this definition.

An important rule when customizing processes to meet the needs of a specific project is not to alter the semantic meaning of existing activities. These activities have been defined and uniquely identified within the Industry Models, and may be reused in a number of different business contents. Altering the meaning of that activity within one or many processes will adversely affect the integrity of the models. For example, an activity should always produce the consistent output triggers in any process context. If conflicting outputs are required, then a new activity should be specified.

Where an entirely new activity is required, care must be taken to define it fully and correctly, based on the standard Industry Model naming conventions and other standards. In particular Industry Model standard verb and nouns should be used as defined in the foundation models. This will greatly assist in the later identification of reuse, by removing some of the natural ambiguity of language, and allowing analysts to search for activities and processes under standard names.

Control Flows

Processes are constructed from reusable activities connected together in a logical control flow. Analysis of these control flows is an essential part of process customization.

All potential endings including drop-outs of the process must be analyzed. Are there positive, neutral or negative control paths that have not been considered? Are there opportunities to detect unfavorable circumstances early on in the process flow, and to abort and save on expensive processing? Drop out cases from a process should be identified as early as possible in the process timeline in order to prevent expensive and time consuming processing of cases that are unlikely to, or known not to be capable of, running to completion.

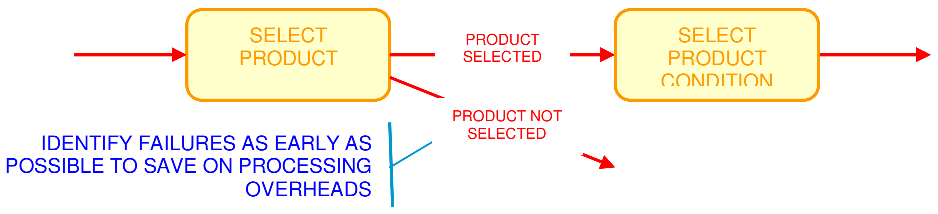


Figure 14. Example drop out in control flow

Similarly there will be opportunities within a control flow for parallel processing. Where there are no dependencies between two work streams, they can be modeled as two parallel threads that may or may not be actually executed in parallel at runtime.

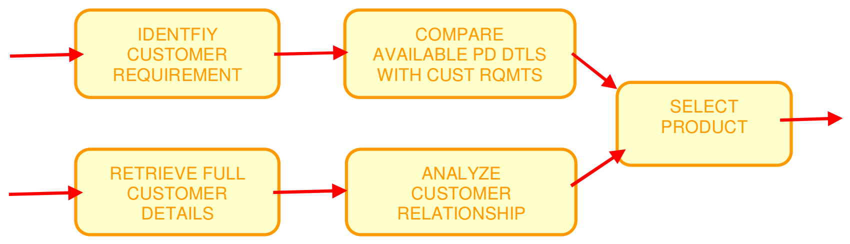

IDENTFIY CUSTOMER REQUIREMENT
COMPARE AVAILABLE PD DTLS WITH CUST RQMTS
SELECT PRODUCT
RETRIEVE FULL CUSTOMER DETAILS
ANALYZE CUSTOMER RELATIONSHIP

Figure 15. Example of parallel work streams

Parallel work streams can only be used where there is no data or trigger dependency between the two streams. Where a dependency exists between flows this dependency must be clearly modeled.

Control flows bring together the Initiating (Input) and Completing (Output) Trigger Logic analyzed earlier. Activities are related in the order of events with inputs and outputs to form a process. Connecting the completing logic of one activity to the initiating logic of the next activity results in a control flow for a process.

Data Flows

Once a stable control flow has been established for the process, analysis can begin of the data that flows along each control path. At analysis time, this begins with a simple business identification of the concepts that flow through the process. These business concepts are added as simple name and definition pair, and used to identify what data requirements exist within the process, in terms of the standard data definitions of the Industry Models.

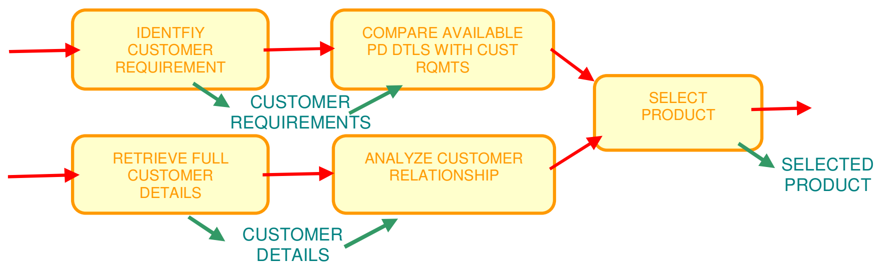

IDENTFIY CUSTOMER REQUIREMENT
COMPARE AVAILABLE PD DTLS WITH CUST RQMTS
CUSTOMER REQUIREMENTS
SELECT PRODUCT
SELECTED PRODUCT
RETRIEVE FULL CUSTOMER DETAILS
ANALYZE CUSTOMER RELATIONSHIP
CUSTOMER DETAILS

Figure 16. Example of analysis data flows

Where detailed business use case documents are available, data flows can be deduced from the inputs and outputs specified in these. Note that the relationship between the data flows within a process and the parameters of the underlying business services is not one-to-one. Contextual information will be passed along the flow that has no relevance to a candidate business service. Similarly, not all the information retrieved by calls to candidate services will be required in a given specific business context, and thus need not necessarily be passed along the flow. This means that the data flows within an analysis process will not provide a direct copy of the parameters required to describe a candidate service, and vice versa.

In order to maximize consistency across modeling domains, the data definitions used to support data flows should be sourced where possible from the standard Industry Model data definitions within the foundation models (e.g. IAA Business Terms or IFW FSDM), preferably expressed as a business glossary (See 3.8.1 "Develop Business Glossary"). Consistency of data definitions across the enterprise can be ensured by mapping the analysis level data flows back to the data definitions that support them within the foundation models or glossary. This will assist both in the provision of detailed data definitions during use case analysis, and during the design of runtime workflows. This will also allow for capabilities such as traceability and impact analysis later in the development lifecycle.

Manual and Automated Activities

When the analysis of the activities, triggers, data and overall control flow has been completed, each activity may be considered to determine the pattern of behavior that it represents. For example, activities may represent human behavior, decisions or rules, analytics, information access or portions of business logic. This information is useful when identifying the form of service that an activity may represent later in the method.

Standard activity naming can act as a guide in this task. Certain standard Industry Model verbs typically indicate particular forms of activities. For examples verbs such as "Consult", Define and Negotiate tend to indicate human intervention in a process. In this way, using verbs with standard definitions makes it easier to identify activity patterns for service identification.

Where an activity is designated to be automated, it is important that any further decomposition of that activity supports this assertion. If, while decomposing an activity, it is later determined to have manual or front-end intervention, then that activity cannot be a candidate service use case.

Roles/Flow Bands

Further analysis of the roles and skills involved in a process will lead to the assignment of activities to Flow Bands. Flow bands are horizontal or vertical bands within a process that identify the role that executes that part of the flow.

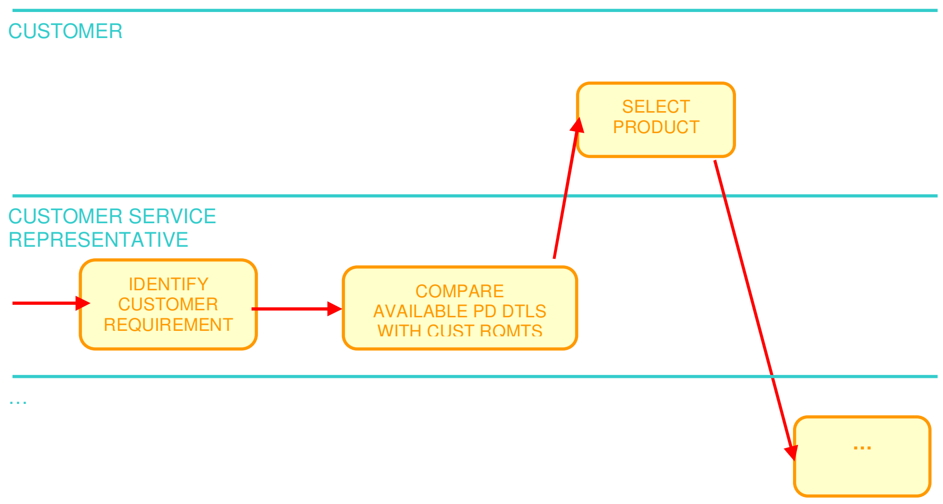

CUSTOMER
SELECT PRODUCT
CUSTOMER SERVICE REPRESENTATIVE
IDENTIFY CUSTOMER REQUIREMENT
COMPARE AVAILABLE PD DTLS WITH CUST RQMTS
...
...

Figure 17. Flow bands within a process

The use of flow bands is an ideal mechanism to capture the separation between tasks that are performed internally and externally to the organization, as well as system tasks and the tasks performed by specific roles within the organization.

Identifying Reuse

Standard naming conventions are key to the identification of reuse across the enterprise. Without the use of standard verbs and nouns, potential reuse becomes clouded by the use of differing lexicons by different analysts. It is only through the use of agreed and understood naming conventions and definitions that multiple analysts, operating across multiple projects, can describe a set of process patterns in a way that can be clearly and unambiguously understood by all.

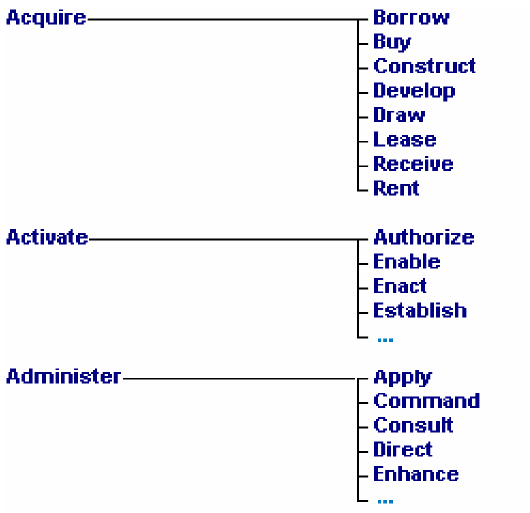


Figure 18. Standard Industry Model verbs

When attempting to identify reuse across multiple business processes it is not sufficient to rely on a given named element occurring in multiple flows. Definitions must be precise and detailed enough to identify that the recurring activity is actually the same pattern. If definitions are not sufficiently detailed, the actual intent of an activity becomes ambiguous, and that activity can be reused across multiple processes, with subtle changes in business meaning in each context. This is not reuse of an activity, it is a 'false' reuse caused by poor activity definitions. Similarly, looking at the informational requirements (inputs and outputs) and resultant triggers across multiple business processes should give a consistent picture of an activity, not one that is tightly bound to its business context.

Recursive Process Decomposition

When customizing processes, it becomes clear that process elements may be recursively decomposed to a very fine grained level. While in many cases this provides essential detail of the requirements, in other cases it can represent a huge analysis overhead with little return on investment. The key question is: when is a process decomposed to a sufficiently fine-grained level? The purpose of customizing the Industry Models is to capture business requirements. This exercise is useful because it helps the organization provide essential guidance to design and development in regards to the solutions that must be constructed to support the business. It is this mission that dictates when process decomposition should stop. When a process has been decomposed to the point that it becomes prescriptive about how the solution to a given requirement should be designed, then that process has been decomposed too far.

Multiple threads of analysis / design

As each business process passes finalization and review, that process can be used to provide a stable platform for the ongoing analysis that is required to support the project in hand. This document places a particular focus on the analysis and design of reusable services, which are a subset of the total requirements expressed by most business processes. It is assumed that manual procedure development, rules analysis, front-end application modeling and design and other analysis/design threads are also occurring based on the analyzed business processes. It is important to understand this relationship between business processes and the analysis design threads that support discrete parts of the process. It is by looking at a business process that we often see the unified content for requirements that impact not just the process itself, but the constituent parts which are invoked as part of that process at runtime,

This does not imply that each of these analysis/design threads is in fact a pure top town driven exercise. Any of these threads may be influenced to varying degrees by bottom up concerns. For example we may identify within a business process a recurring data access requirement, implying an information service. However, we may choose to support this information service almost entirely from a bottom up perspective – perhaps by exposing a federated query as a service. This will usually result in a compromise between the pure requirements expressed during process analysis and the resulting invocation from within our runtime process. Similar compromises may occur within rule based requirements, business logic requirements, or any other part of the analysis process, however it is preferable to delay these compromised until design time, allowing the analysis model to retain a pure expression of business requirements.

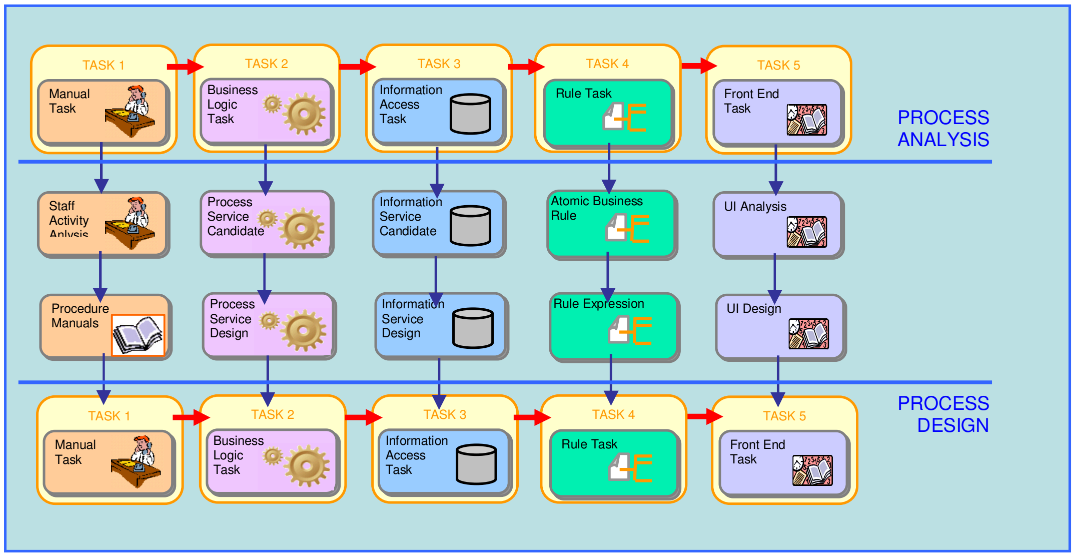


Figure 19. Multiple threads of analysis and design based on a process

Delegation Patterns

Business activities that represent candidate services can now be formally identified and either scoped into a view or project scope to track their intent, or mapped to newly created use cases. Note that these activities can be quite complex in their nature. In many cases an invocation to a service will be the start of a complex collaboration of services to meet a business need. At this stage it is not necessarily relevant how the service is supported, just that service candidates be identified.

Service candidates are not necessarily found at the leaf level of process analysis either. In many cases, whole sub-strings of process will be automatable end to end, and will represent valid service candidates. In these cases, the decomposition below the service candidate should be viewed as a quick start to the internal view of the use case describing the service candidate in question.

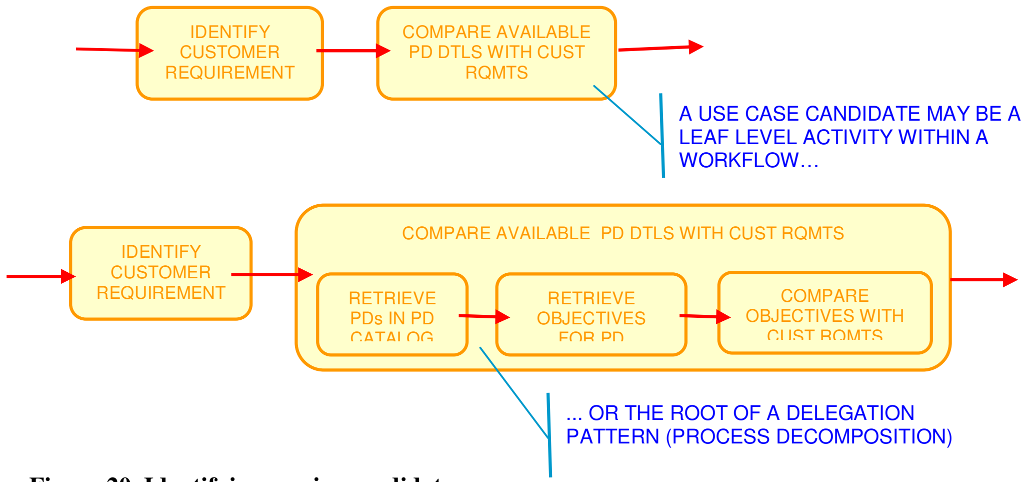

IDENTIFY CUSTOMER REQUIREMENT
COMPARE AVAILABLE PD DTLS WITH CUST RQMTS
A USE CASE CANDIDATE MAY BE A LEAF LEVEL ACTIVITY WITHIN A WORKFLOW…
IDENTIFY CUSTOMER REQUIREMENT
COMPARE AVAILABLE PD DTLS WITH CUST RQMTS
RETRIEVE PDs IN PD CATALOG
RETRIEVE OBJECTIVES FOR PD
COMPARE OBJECTIVES WITH CUST RQMTS
... OR THE ROOT OF A DELEGATION PATTERN (PROCESS DECOMPOSITION)

Figure 20. Identifying service candidates

An activity within a business process may have a one-to-one mapping to a service, or represent a superset of a service, or indeed a set of services. That is to say that a given activity may be equal to or coarser grained than the underlying services. A given service definition should never be fragmented across multiple activities within the process.

The identification of these delegation patterns between service candidates is an early hint of the delegation patterns that will occur between services at design time, and is an essential pattern within reusable service architectures.

Verifying Context of Use

When a service candidate has been identified, it is important to verify that it represents truly a reusable service. This requires an examination of each known business context of the use case. Where is the activity occurring within a business process? Is the semantic meaning of this activity consistent across all these business contexts? If a business activity exists in multiple business processes, but is not consistent in its interpretation across those processes, then it is actually representing many different activities that have been given the same name. This is traditionally a very difficult piece of analysis to support – however the delivery of IBMs Industry Models with reused activities across multiple business contexts enables this.

Accurate business definitions are essential to verifying context of service candidates. Only if the business intent is clearly and unambiguously defined can the consistency of that intent be verified.
Where the same activity is used inconsistently in many processes that activity should be split up into several activities that more accurately represent the business intent of the process. In many cases the decomposition of these activities is where the real reuse can be identified.

The data pattern of reused activities should also be consistent across business contexts. This relates to the expressed informational requirements of that activity. The input and output informational requirements do not necessarily have to be identical, as some allowance must be made for optional parameters, however, a consistent data pattern should be identifiable.

Selecting and Prioritizing Service Candidates

Service candidates can be prioritized according to the degree to which they are reused. Activities that are reused across many business contexts are important service candidates and should get priority because the potential benefits of implementing that use case are high. Key services that are highly reused in a very consistent way across the enterprise should often be the first target in service architecture, as they represent a quick win for the organization as a whole. However it is also important to consider the technical implications of supporting a service. A service candidate that is highly reused, yet extremely difficult to support is of limited value. Many of these design constraints around service candidates are not apparent until service design. This means that service candidates that are prioritized during analysis may later be descoped or reduced in priority during design.

Missed Reuse Opportunities

While identifying use cases within the customized processes, cases where opportunities for reuse have been missed will be encountered. This may be because of a deviation from standard naming conventions, a difference in the level of abstraction, or simply a misinterpretation of requirements.

Naming standards help greatly in the identification of reuse, but it is impossible to guarantee that two different analysts will use the same terms to define a given requirement.

Similarly, a given requirement can be defined at different levels of abstraction, from the very generic to the highly specific. Defining the right level of generalization for an activity is a balance between two opposing forces. The force to make definitions generic and thus reusable and the force to make definitions specific to retain business applicability.

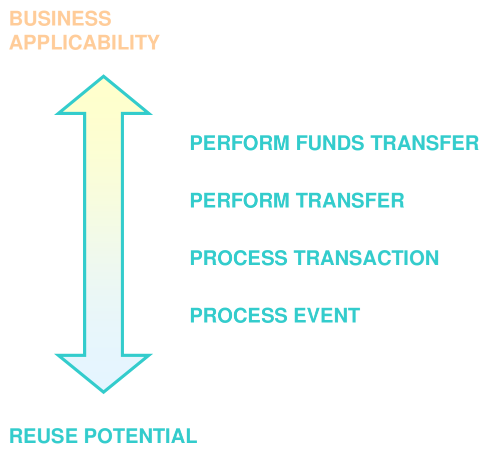


Figure 21. Spectrum of generalization

There is no guarantee that the same level will be selected by all analysts in all contexts. In many cases it can be important to retain the specific (and different) definitions of activities, and identify the reuse potential through a generic activity within that business specific activity.

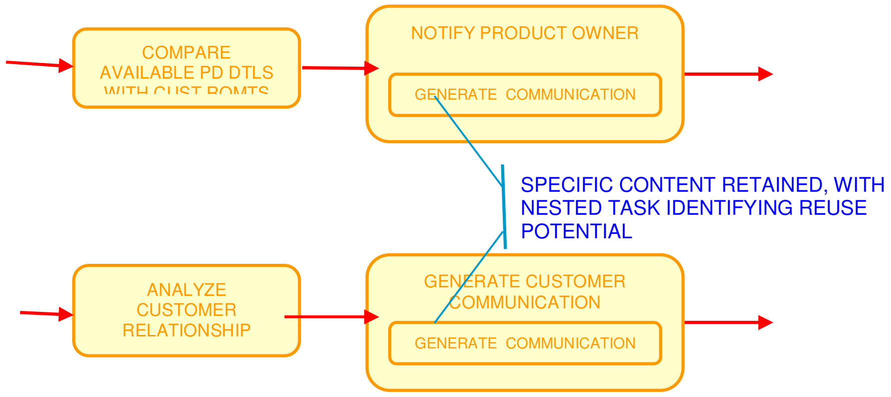

COMPARE AVAILABLE PD DTLS WITH CUST RQMTS
NOTIFY PRODUCT OWNER
GENERATE COMMUNICATION
SPECIFIC CONTENT RETAINED, WITH NESTED TASK IDENTIFYING REUSE POTENTIAL
ANALYZE CUSTOMER RELATIONSHIP
GENERATE CUSTOMER COMMUNICATION
GENERATE COMMUNICATION

Figure 22. Reuse within differing contexts

This results in a parent activity that maintains the business applicability of the task, and delegates to a more generic reusable activity that supports the essence of the pattern involved. In either case, it is essential that use case identification is not mechanical. Human interpretation is essential to maximizing reuse across the organization.

3.4.3. Validate Business Processes

The processes that are produced during the initial customization should be reviewed in the light of business requirements and stated organizational strategy before being finalized. In particular, it should be verified that the modeled processes meet business objectives for the project, that they observe good engineering principles, and that they adequately capture the business requirements involved.

Verifying Business Requirements

By matching the customized business processes against the stated requirements and strategic objectives, determine that these requirements and objectives have been correctly and fully captured. It is essential that the requirements that are captured within the processes are clear and not open to interpretation. Definitions should be complete and no ambiguity should exist in the stated purpose or scope of any business activity. Business use case documents should be available for the customized processes, and should in particular include business rules and informational requirements.

Completeness of Scope

It is also possible that the reviewed processes accurately and completely capture the requirements within their scope, but that additional processes are now required to cover the full scope of the project in hand. This does not necessarily imply scope creep. It

may only be possible to clearly see the “missing” scope elements once the processes have been fully customized.
Any additional processes that are scoped at this stage should represent a very minor percentage of the full project scope. If not, the original process scoping should be questioned. The functional scope for the project should be used to determine the completeness, as described under “Functional Area Analysis”.

Non-functional Requirements
The customized processes should also be reviewed in the light of several non-functional requirements. Processes must be capable of performing appropriately. This should include factors such as key performance indicators (e.g. cost, time, quality, volumes, turnaround time, response time, service level agreements and key dependencies).

Have all other non-functional requirements been captured and are those non-functional requirements still reasonable in the light of the customized process?

Stabilizing Processes
The customized and finalized processes for the project will form the backbone for the subsequent key analysis areas in the project lifecycle. The logical process model identifies the presence of manual tasks and front-end application requirements, as well as service candidates.

It is essential that the process be fully reviewed and agreed before each of these separate concerns takes the analysis to the next stage. Two key elements in validating these processes are initial use case identification, and process simulation.

Initial use case analysis involves sweeping the business processes to identify service candidate use cases amongst the process activities. This does not involve creating or modeling the actual use cases. The goal here is to identify whether the granularity of the activities is correct, and whether there are activities that should be rolled together or separated.

Where an activity implies a service candidate, it should be possible to clearly identify that activity independently from its surrounding business activities. If not, it is likely that the activity in question is not sufficiently decomposed. One further layer of decomposition of that activity will usually result in the separation of the service candidate from the surrounding logic.

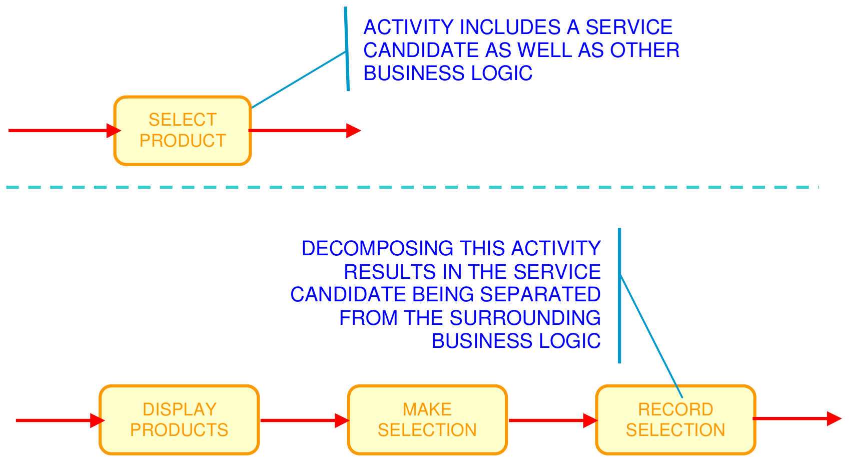

ACTIVITY INCLUDES A SERVICE CANDIDATE AS WELL AS OTHER BUSINESS LOGIC
SELECT PRODUCT
DECOMPOSING THIS ACTIVITY RESULTS IN THE SERVICE CANDIDATE BEING SEPARATED FROM THE SURROUNDING BUSINESS LOGIC
DISPLAY PRODUCTS
MAKE SELECTION
RECORD SELECTION

Figure 23. Decomposing business activities

Sweeping the process to identify use case candidates may also identify cases where a number of business activities have been expressed separately during process analysis, when in fact they can be performed by a single front-to-back-end delegation or service candidate. In these cases, the activities can be rolled up into a single business activity. If detailed business collaboration occurs between services, within that service candidate, that information can be captured in the internal view of the use case during object analysis in BOM or in the "main flow" section of the use case template.

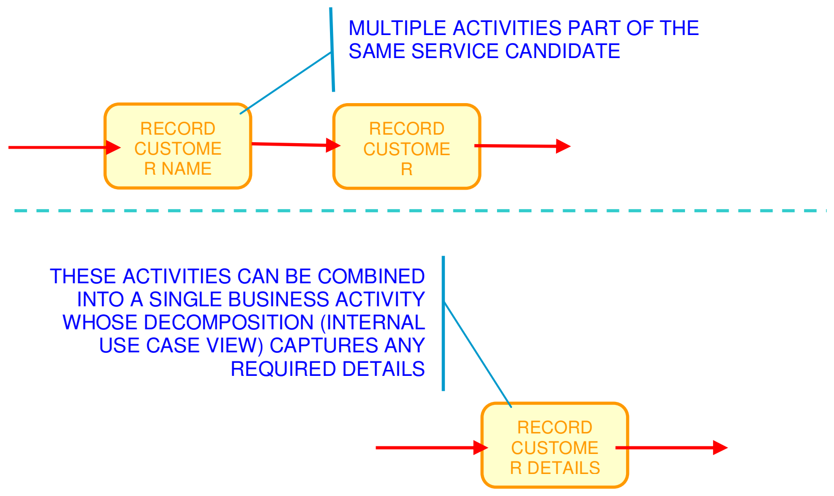

MULTIPLE ACTIVITIES PART OF THE SAME SERVICE CANDIDATE
RECORD CUSTOME R NAME
RECORD CUSTOME R
THESE ACTIVITIES CAN BE COMBINED INTO A SINGLE BUSINESS ACTIVITY WHOSE DECOMPOSITION (INTERNAL USE CASE VIEW) CAPTURES ANY REQUIRED DETAILS
RECORD CUSTOME R DETAILS

Figure 24. Combining business activities

Process Sign-off
At this point in the project, it is recommended that the finalization of the customized processes be signed off as accepted by business and architecture and recorded as a key milestone completed.

Simulating Processes
In addition to a first pass of use case identification, a phase of simulation and costing may be included as part of the final review of the customized processes. There are software environments such as IBM WB Modeler that support the evaluation of time and cost measures for a business process through simulation and optimization. During this phase of analysis, additional changes may be made to the overall business process to enhance its scalability and performance.

It is important that this evaluation is done before the process is used to influence the analysis and design of reusable business services, manual procedures, and front-end applications.

Once the business process has been finalized and optimized based on use case analysis and process simulation / costing, it may require a further check (and possibly sign-off) by the business process team, to satisfy them that the changes have not altered the fundamental behavior and requirements of the process.

3.5. Identifying Service Patterns

As highlighted in “Figure 19. Multiple threads of analysis and design based on a process”, a business process shows the common context for a number of service patterns. This section highlights the differences between these patterns and the indicators that a particular activity in a business process is of a specific pattern. The resulting services of each pattern will tend to be distributed across the component architecture, as opposed to grouped with services of that specific pattern. Note that this discussion involves a certain degree of looking ahead to the design time constraints implied by each service pattern. This is not intended to imply that these constraints should be applied during analysis time.

3.5.1. Process Services

Process services are services that encapsulate meaningful business logic, allowing that logic to be reused across the enterprise. Process services are characterized by moderate to low data access and a reliance on information that has been retrieved earlier in the process context. Process services tend to operate on this information to derive or calculate additional information that is of value to subsequent steps in a process. Process services will tend to originate from business activities with the standard root verbs “Decide”, “Activate” and “Administer, and will typically be coarse grained and will delegate to other services through a service collaboration.

Process services identified from business activities will be further analyzed through BOM use cases, resulting in IDM services which define coarse grained business logic, leveraging a number of finer grained services, implemented through traditional programmatic means or through a collaborative standard such as BPEL.

3.5.2. Rule or Decision Services

Rule or Decision services provide a means to expose business rules into services architecture, allowing them to be consistently reused. Rule or Decision services are typically finer grained than process services, although complex rule or decision services may be constructed through a collaboration of multiple fine grained rules and decisions. Rule or Decision services are frequently reused as part of process services, rather than being exposed to the business process layer directly. The analysis and design approach of rule or decision services may differ from that of process services due to the fact that they are typically externalized behavior supported by a rules engine. Section 3.6 “Record Business Rules and Policies” represents the root of the analysis and design of rule or decision services, highlighting the means through which the rules encapsulated by rule or decision services are identified. Section 3.10 “Rule and Policy Analysis” further refines the logic of these decision services. Ultimately access to rule or decision services is structured in the same was as that of process and information services, so they still result in BOM use cases and IDM service elements. The key difference is that the logic within a rule or decision service is externalized from the point of view of the Industry Models.

Section 3.9 “Variation Oriented analysis” is key to the structured analysis of business rules, identifying the business variations that are suitable for encapsulation as decision services, and their relationship to other elements within the Industry Models,

3.5.3. Information Services

Information Services expose structured and unstructured information access to service consumers providing trusted and accurate data access in order to address the following requirements:

- Access to master data occurs across many different business contexts. Each of these contexts has a particular view of the specific data representations they require. Information services seek to provide context independent access to data through analysis of the most reusable data patterns across the enterprise. This prevents a proliferation of services that access information, each expressed from the point of view of a single business context.

- Much of the information required by service consumers does not reside in a single data source. In many cases complex information integration patterns are required to provide access to information. Information services hide the complexity of this data integration from service consumers.

- Information services allow controlled access to data. Information services follow the same security principles as any other service, allowing control and governance of information access.

Information services provide a portal into Information On Demand, supporting data cleansing, transformation, analytics and integration, without exposing service consumers to the complexity of this environment.

Information services may be identified through the analysis of activities of the root verb "Manipulate". Candidate activities will retrieve information used elsewhere in the business process context, or record information that has been captured during the process. In particular, access to master data (Customer, Involved Party, Product, Arrangement) usually implies an information service. In addition, services that access unstructured information or services that rely on content management are key candidates.

Information services can rarely be analyzed through a purely top down method. While identification of the most reusable information access and data structures is critical to the analysis of information, due consideration must be given to the persistence of this data and the impact of extensive runtime transformation. Analysis and design of information services requires a careful balance between the expression of data that is most useful to service consumers, and that which the information platform can realistically support. Information services may be explored through the use of BOM use cases to drive a detailed understanding of the information requirements of service consumers, however at design time a mapping of these structures to the structures that are available from the data platform may yield significant compromises.

3.6. Record Business Rules and Policies

Business rules will manifest in a variety of locations within a model set, particularly within process analysis. Certain forms of rules will be implicit in the definition of the business types in the model, and will manifest in UML and entity relationship (ER) models of the business types of the modeled domain. Other forms of rules will detail the allowed changes in these business types, manifesting as constraints within service definitions, and state transition models. We will also encounter rules that affect the flow control of business logic. These rules will usually be encountered within process or service flow control, within the process and service analysis models.

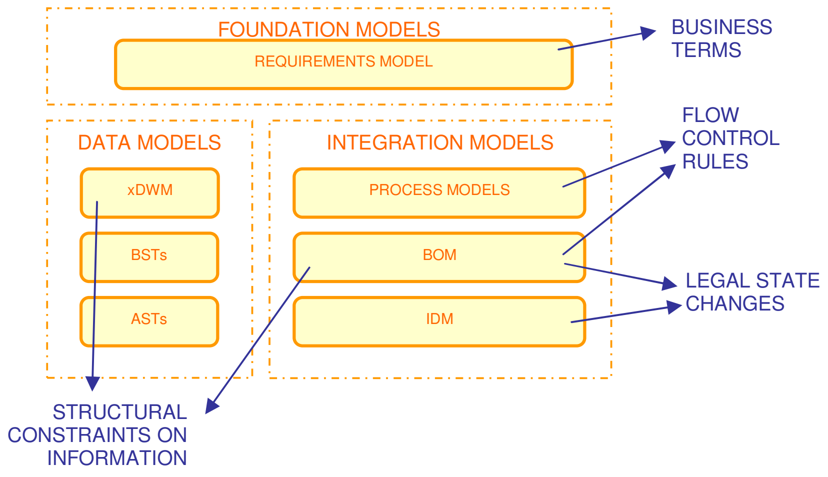


Figure 25. Sources of rules within the Industry Models

This gives rise to a defined set of rules origination patterns – parts of the Industry Models where rules tend to arise and require specification:

- **Rules relating to business processes or activities**
  These rules relate to flow control logic within a business process model, affecting the branching, and flow control of a sequence of business tasks. Rules of this nature tend to arise at analysis time within process models and require formalization to drive soft coded process flow (e.g.: BPEL) or application logic.**Rules relating to services**
  These rules relate to the execution of service logic and relate closely to rules relating to business processes and activities. As a consequence, these rules often also arise during process analysis, however they will require further analysis as part of the business object model, before being formalized as part of service specifications within IDM.

- **Rules relating to information**
  These rules relate to information, relationships between information, and the permitted states of information. Rules of this nature will arise during informational analysis, either as part of the business object model or logical data modeling. These rules may be formalized within service specifications to assert control over information structures, or within the information platform itself.

All of these rule types must be identified, analyzed and designed in a structured way, maintaining their relationship to the model artifacts in which they originated, and the model artifacts which they reference or operate on. This requires a defined method to support the analysis and design of business rules, a defined set of formats in which these rules can be captured, and formalized relationships between these rules and other model artifacts.

We care about capturing and managing these rules because they are an integral part of the requirements definition, analysis and design of software systems – irrespective of whether these rules are ultimately externalized to a rules engine, hard coded in software, or implemented through human procedures. These rules compliment the data structures, processes and services of the Industry Models, allowing a complete picture of business requirements to be expressed.

3.6.1. What is a business rule?

A business rule is a statement that defines or constrains some aspect of a business. The intent of a business rule is to control or influence some aspect of a business thorough the imposition of structure.

Forms of business rule

The term ‘business rule’ is extremely broad, and requires further classification. For the purpose of this document we will deal with four distinct forms of business rule.

Business Terms

A *business term* is the most fundamental form of business rule. Through the definition of a term, we are formalizing how stakeholders communicate about a concept. Terms are traditionally captured with a business glossary, or as concepts within a model (such as an ER model or class model).

An example of a business term is "*customer*" – defined for our purposes as "A role played by a party that is considered to be receiving services or products from the modeled organization or one of its organization units, or who is a potential recipient of such services or products".

Facts

The structure of a business can be described in terms of *facts* which relate business terms to each other. Facts may take the form of natural language statements or as attribution, relationships or generalizations within a formal model.

An example of a fact is *"a customer has an address'*.

Derivations

Derivations describe how information in one form may be transformed into information in another form – through an inference or mathematical calculation. An inference produces a derived fact using logical induction or deduction. A mathematical calculation produces a derived fact according to a specified mathematical algorithm.

Examples of derivations are *"two customers with the same address, first name, last name and date of birth are the same customer"* (inference) or *"average customer profitability is a summation of all customer profitability divided by the number of customer profitability"* (mathematical calculation).

Constraints

While facts describe relationships between types within the business domain, *constraints* describe limitations in that domain. Constraints act upon a business term (a subject), through a correspondent rule – usually a fact. A correspondent rule is the rule that enables a constraint.

An example of a constraint is: *"a customer must have exactly one legal address"*. The subject of this constraint is customer. The correspondent rule (fact) is that customers have addresses.

3.6.2. Degrees of formalization

Defining these four forms of business rule allows us to meaningfully discuss what we mean by rules, and how rules relate to each other, however, different participants in the requirements / analysis / design cycle will express rules with varying degrees of formalization and structure.

Business users will tend to express rules extremely informally, usually though the use of natural language, and usually grouping multiple rules into a single statement, for example *"…potential customers must be credit checked and identified…"*. Here, we call these informal definitions Business Rule Statements. Business users will also tend to express themselves through Business Policy, such as *"we only deal with credit checked and verified customers"*. Policies and rules, although closely related, are distinct constructs. Business policies tend to originate and group distinct business rule statements, acting as a general guideline or context to the rules.

Analysts will tend to break rules into atomic rule elements – rule definitions that have been broken down as far as possible without the loss of business information. A given business rule statement will often result in multiple interrelated atomic business rules. While still expressed in form of natural language, atomic business rules tend to be more formalized and precise, and will normalize a rule definition, eliminating overlap between related rule statements.

Rule designers can then take these atomic rules and express them in a format specific to the target environment. Where a rules engine is to be used, this format will usually be specific to this rules engine. Note that a single atomic business rule may manifest as a number of formal rule statements, each with a different formal expression type. It is not necessary that this expression be related to an externalized rule execution environment or rules engine – rules may often be expressed "hard-coded" into application logic, or indeed executed by staff within an organization.

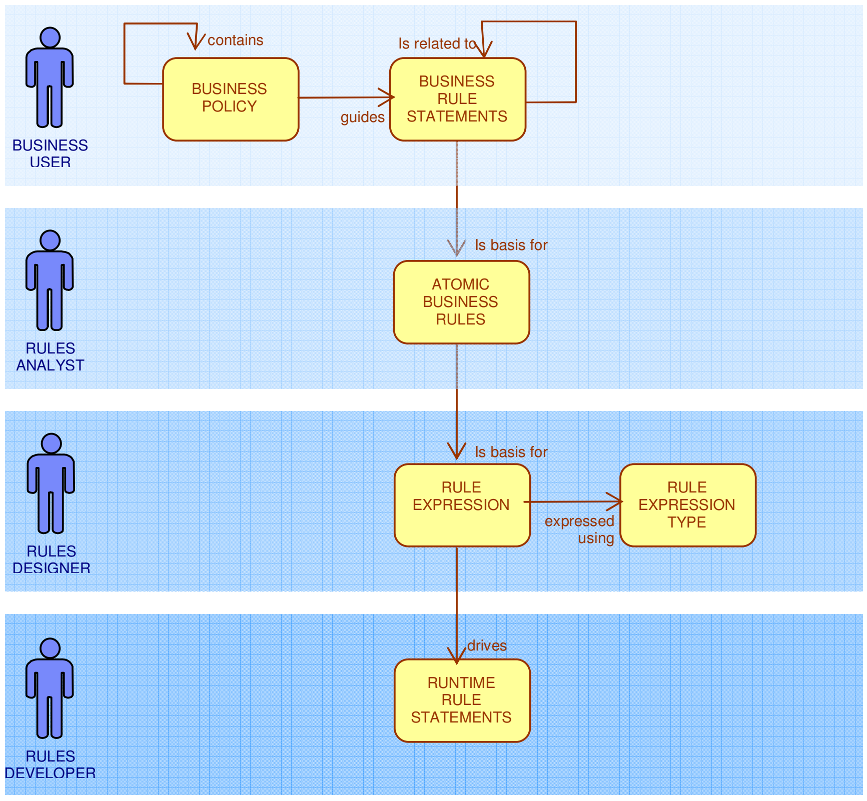
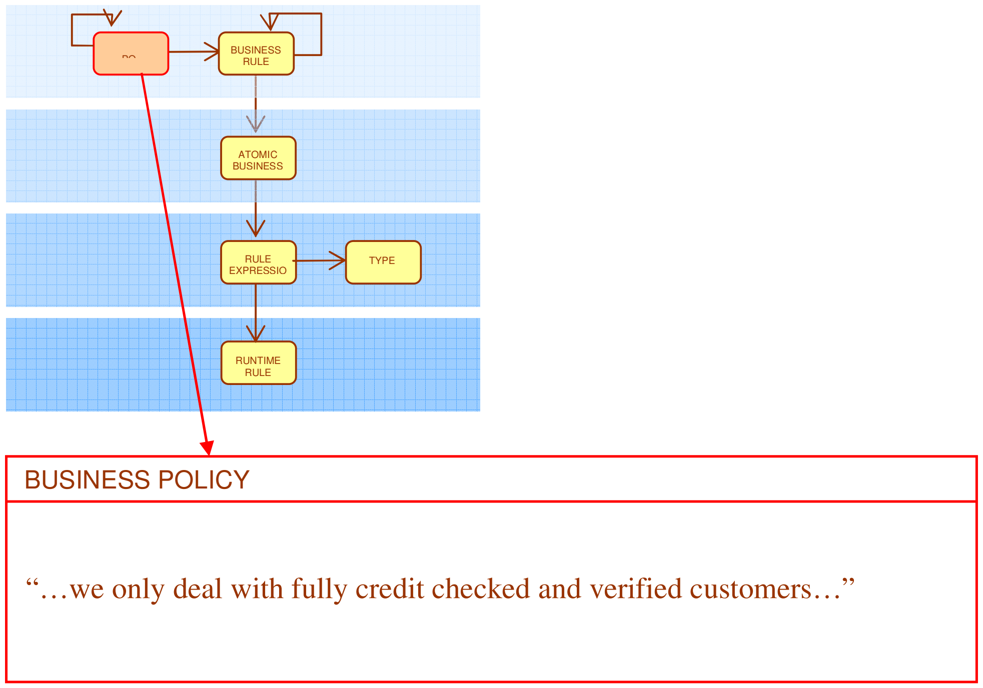


Figure 26. Degrees of formalizing business rules

The goal of this portion of SOMA is the identification and documentation of the relevant rule and policy statements that are required to fully express business requirements. Further formalization of these requirements to define atomic business rules will occur during section 3.10 "Rule and Policy Analysis"

3.6.3. Business policies

Business policies are informal expressions of business direction, usually formulated by business users to guide the business as a whole. These policy definitions tend to appear at an early stage in the analysis design lifecycle, and are often expressed as additional detail during business process analysis. Policies themselves rarely manifest as business rules, they tend to act as the guiding principles within which rules are expressed.



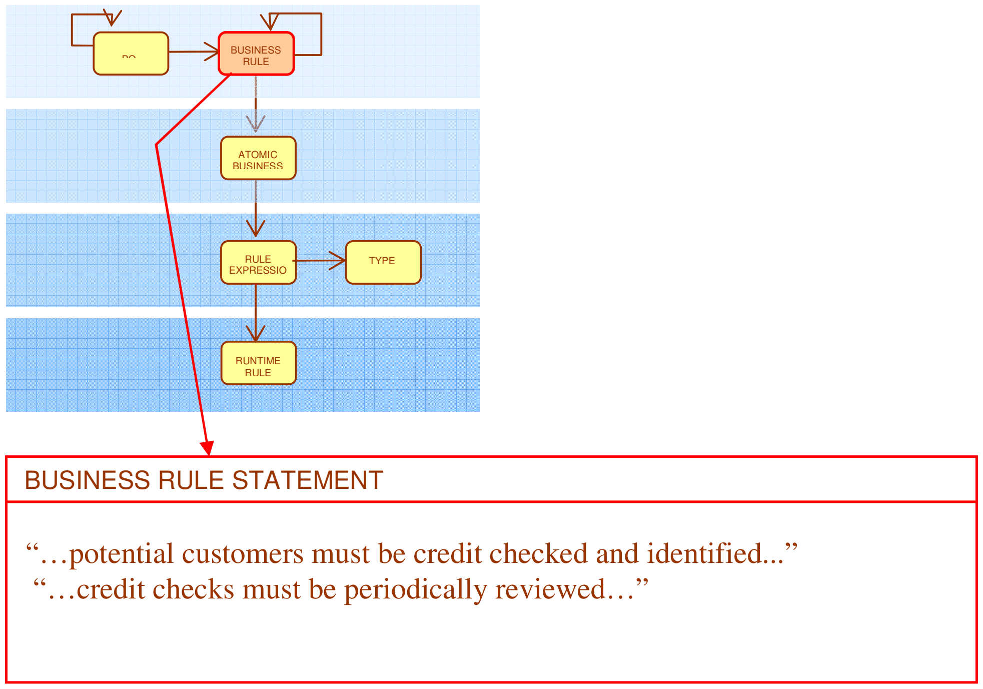
"…we only deal with fully credit checked and verified customers…"

Figure 27. Example business policy

3.6.4. Business rule statements

Business rule statements provide the additional detail necessary to enforce a business policy. These are also informal and unstructured statements, expressed directly by business stakeholders. Business rule statements will often contain multiple discrete instructions that are closely related, and it is often non trivial to break business rule statements into atomic business rules.

Business rule statements are usually closely related to activities within a business process, and it should be possible to identify the activity or activities within a process where a set of business rule statements are relevant. Doing so will help to ensure a clean linkage between this activity in a business process, and the rules that are ultimately required to support it.



"…potential customers must be credit checked and identified..."
"…credit checks must be periodically reviewed…"

Figure 28. Example business rule statements

3.6.5. Capturing policies and rule statements

Section 3.4.2 "Process Decomposition" highlights the benefits of capturing textual requirements definitions during process analysis. An example template for these requirements is provided, a subset of which is shown below.

Business Rules	A definition of any specific rules pertaining to this activity. These are often provided as a bulleted list, or "if…then" conditional statements that describe how the activity is to be executed.
Critical Success Factors	A definition of any preconditions or success factors that must exist for this activity to execute successfully. These are often provided as a bulleted list. This may include a description of innovated areas that will be done differently in the new process.
Constraints	A definition of any known limitations or restrictions on the process. This may include legal or regulatory requirements as they pertain to this activity.
Dependencies	A definition of any logical units or other processes upon which this activity will depend for its operations. This includes policy changes and organizational decisions that must be made in order to implement this activity.
Non-Functional Requirements	Definitions of any supporting requirements that are not directly involved in the business definition of this activity, but still affect the successful outcome of this activity, such as key performance indicators and systems support requirements for the business.

Figure 29. Rules in business use cases

The intent of this template is to encourage the capturing of textual requirements in a semi structured way, such that analysts may later work with them. Documents based on these templates may be created and managed alongside the formal models, providing the requirements input needed to drive analysis and design threads.

Management of these semi structured requirements may be accomplished in a number of environments, enabling structured requirements documents to be mapped to model elements. This allows business policy and rule statement information to be captured in textual form and mapped directly to the business activities and UML structures to which they relate.

Relationships may also be established between these requirements types. For example Business policies may be linked to the business rule statements that support those policies. Similarly these requirements can be related to the business model elements where they are relevant. For example, in a banking context, the business policy "…we only deal with fully credit checked and verified customers…" could be expressed as a requirements document, mapped to the business process "Provide Banking Product Arrangement Offer", which deals with taking a documented set of customer needs and producing an offer to meet those needs. In the same way, the business rule statement "…potential customers must be credit checked and identified..." could be mapped to the activity Analyze Customer Relationship within this same business process.

This provides a means to capture not just the simplified examples presented here, but complex policy and rule statements, linked to the business contexts in which they apply.

3.7. Identify Candidate Services

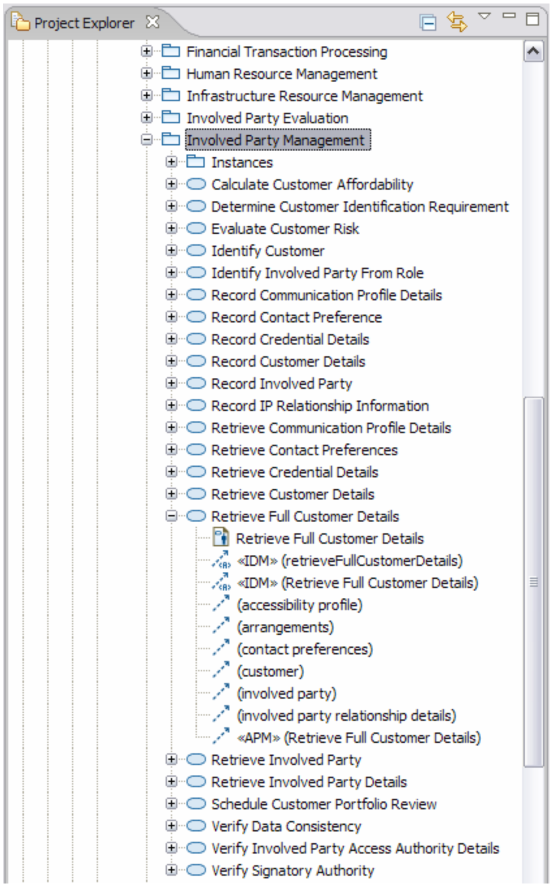
Each identified service use case within the customized processes must be formally defined as a service candidate within the BOM model. This involves locating an existing use case, or creating a new use case, modeling its information requirements, analyzing its decomposition where required, and modifying any type information within the model as needed to fully capture requirements.

While most of the patterns expressed here focus on the analysis of the service contract and its associated data structures, elements of this section (For example internal use case views) may not be required for all forms of service. For example for information services, the degree to which we analyze the ideal service contract will depend on the likely freedom we have to enforce that contract. Where an information service is being constructed by exposing a federated query, and we do not wish to translate the returned data structures into the types expressed by BOM, this analysis can simplify dramatically. However this is a compromise to be avoided where possible, as data transformation then occurs at the process level. For this reason it is recommended to drive the top down identification of all service candidates as far as possible, expressing bottom up compromises at design level. In these cases the service signature (e.g. WSDL) may be enforced by the data platform, and not formed by the IDM model. However the motivation to form service signatures based on business requirements remains, and where the flexibility exists, the steps below remain valid.

Similarly, in the case of decision services, it may not make sense to model the internal use case view beyond a high level definition as provided in a use case document. The analysis and design of the rules that support the decision service may be performed through the analysis of atomic rules, feed into a rules authoring environment. However in either case the analysis of the service contract retains its importance, allowing the identification of the most reusable service signature across business contexts.

3.7.1. *Locating the Use Case*

Use cases in BOM are hierarchically categorized, to allow for structured content management. When searching for a given use case or inserting a use case, this hierarchy allows the analyst to narrow their search according to the business domain in question.. The required use case can then be edited or inserted within this hierarchy.



Project Explorer

- Financial Transaction Processing
- Human Resource Management
- Infrastructure Resource Management
- Involved Party Evaluation
- Involved Party Management
  - Instances
  - Calculate Customer Affordability
  - Determine Customer Identification Requirement
  - Evaluate Customer Risk
  - Identify Customer
  - Identify Involved Party From Role
  - Record Communication Profile Details
  - Record Contact Preference
  - Record Credential Details
  - Record Customer Details
  - Record Involved Party
  - Record IP Relationship Information
  - Retrieve Communication Profile Details
  - Retrieve Contact Preferences
  - Retrieve Credential Details
  - Retrieve Customer Details
  - Retrieve Full Customer Details
    - Retrieve Full Customer Details
    - «IDM» (retrieveFullCustomerDetails)
    - «IDM» (Retrieve Full Customer Details)
    - (accessibility profile)
    - (arrangements)
    - (contact preferences)
    - (customer)
    - (involved party)
    - (involved party relationship details)
    - «APM» (Retrieve Full Customer Details)
  - Retrieve Involved Party
  - Retrieve Involved Party Details
  - Schedule Customer Portfolio Review
  - Verify Data Consistency
  - Verify Involved Party Access Authority Details
  - Verify Signatory Authority

Figure 30. Sample BOM use case hierarchy

3.7.2. *Defining the Use Case*

Having identified the correct location for a use case within BOM, that use case should be created and fully defined.

As described earlier in this book, the correct level of generalization of service candidates is essential to maximize the reuse of the services (use cases). Similarly the correct level of granularity in the use case level is essential to maximize the consumability of the resulting service architecture.

A Use Case has three distinct levels of definition. At the most basic, a use case must have a name and a detailed textual definition. This definition must identify the scope and intent of the use case in question.

The next level of use case definition is an External Use Case View, the function of which is to describe the informational requirements of that service candidate. This External Use Case View provides the analysis of the service signature itself, including its informational requirements. In constructing this view the service candidate should be analyzed across all the known business contexts, and a unified perspective of its required information should be created. In this way, the external use case view acts as a mechanism to analyze the most reusable service candidate from the perspective of many service consumers.

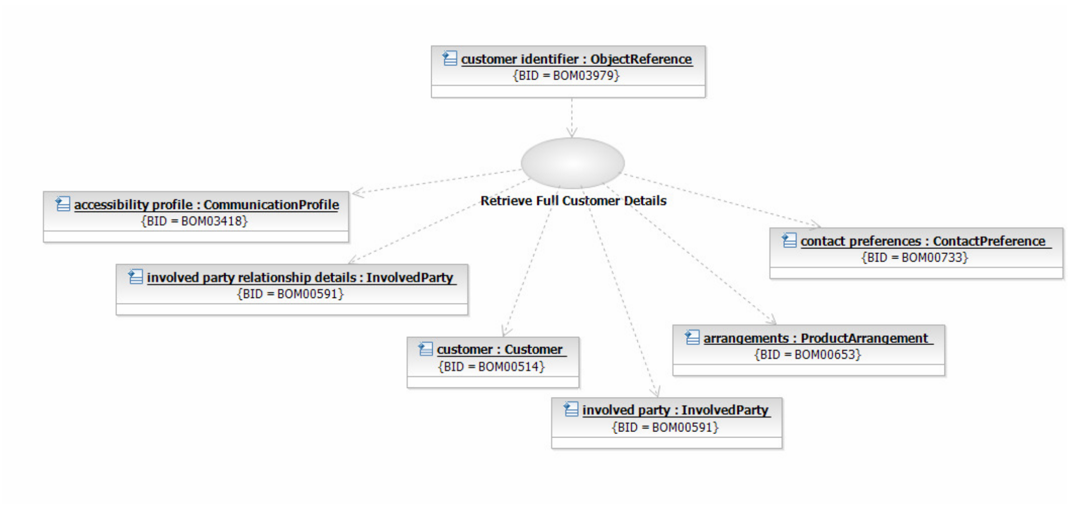

customer identifier : ObjectReference
{BID = BOM03979}
Retrieve Full Customer Details
accessibility profile : CommunicationProfile
{BID = BOM03418}
contact preferences : ContactPreference
{BID = BOM00733}
involved party relationship details : InvolvedParty
{BID = BOM00591}
arrangements : ProductArrangement
{BID = BOM00653}
customer : Customer
{BID = BOM00514}
involved party : InvolvedParty
{BID = BOM00591}

Figure 31. External use case view

This provides a full analysis picture of the service candidate itself, but does not describe the inner workings of that service candidate, or how it delegates to other known requirements. To capture this analysis, the “black box” view of the use case must be opened up to allow a description of the internals of that box. This information is captured as an Internal Use Case View, describing the process flow that must occur during the execution of that service candidate. The key motivation of this step is to identify nested use cases and the relationships between service candidates. As such this step may not be appropriate for all candidate services. In the case of information services, this internal view may be a federated query or other data integration pattern, and is of little value to the analysis of service interdependencies. Similarly, for decision services, this analysis of the rule itself may occur in a separate rules analysis / design activity.

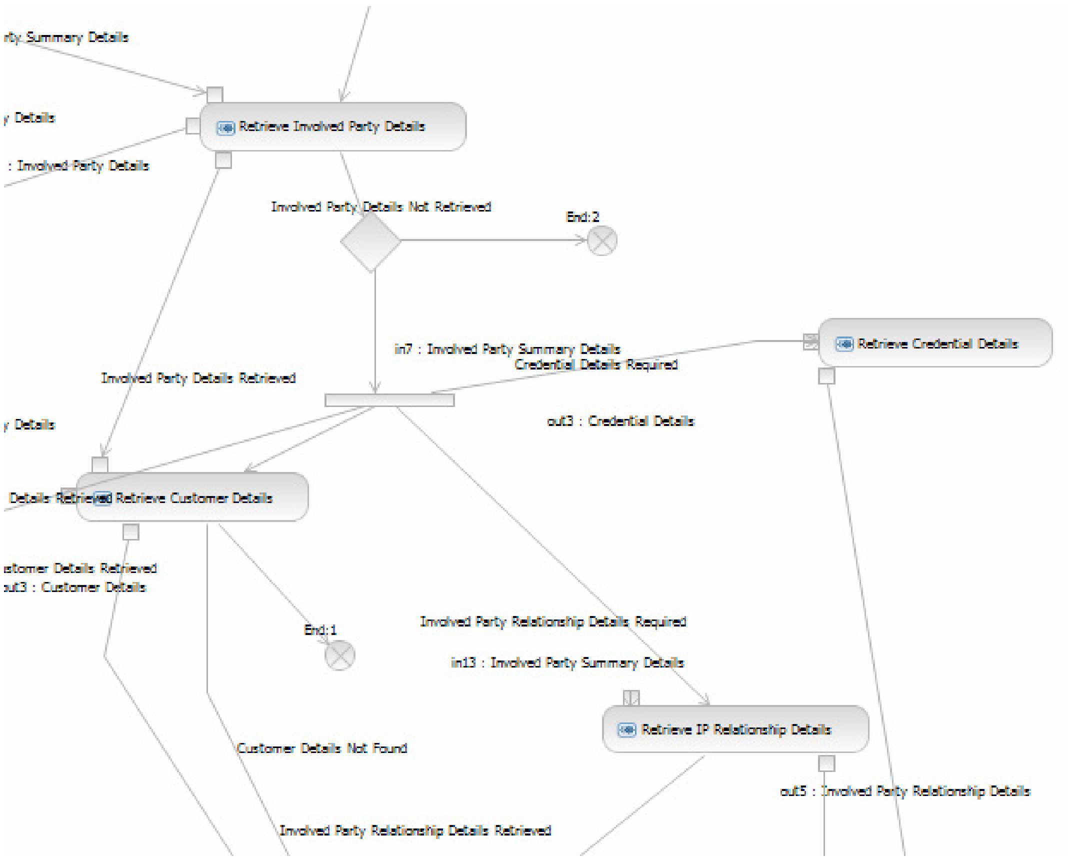

rty Summary Details
y Details
: Involved Party Details
Retrieve Involved Party Details
Involved Party Details Not Retrieved
End:2
in7 : Involved Party Summary Details
Credential Details Required
Retrieve Credential Details
Involved Party Details Retrieved
out3 : Credential Details
y Details
Details Retrieved
Retrieve Customer Details
stomer Details Retrieved
out3 : Customer Details
End:1
Involved Party Relationship Details Required
in13 : Involved Party Summary Details
Retrieve IP Relationship Details
Customer Details Not Found
out5 : Involved Party Relationship Details
Involved Party Relationship Details Retrieved

Figure 32. Portion of internal use case view

This decomposition of the use case should leverage existing use cases definitions within the model or lead to the definition of further use cases where reusable behavior is identified. Each of these use cases will themselves have an external and potentially an internal view. This allows us to build up a clear picture of the interdependencies between use case candidates, and ultimately the interdependencies between the software elements that support those services.

3.7.3. Mapping Use Cases to Process Context

Having located and defined a use case node, the first step is to map that use case to the location or locations within the business process that invoke that use case.

This is a direct mapping between the use case and an equivalent activity within the business process models. Where the use case to activity relationship is not one-to-one, and a business activity involves considerably more than simply calling that use case, a simple decomposition of that use case/activity should be added.

Within this decomposition, an activity that maps directly to this use case can be added and mapped to. In many cases, this activity decomposition needs to be little more than a placeholder for the activity that maps directly to the use case. This is sufficient to capture the analysis that a given service candidate is called at a particular point in a business process.

3.7.4. Adding Informational Requirements

Once a use case has been established and mapped, an external view for that use case is constructed. This involves identifying the informational requirements of the use case. The first step is to add an object instance for each known informational requirement and name it appropriately, providing a business definition where required.

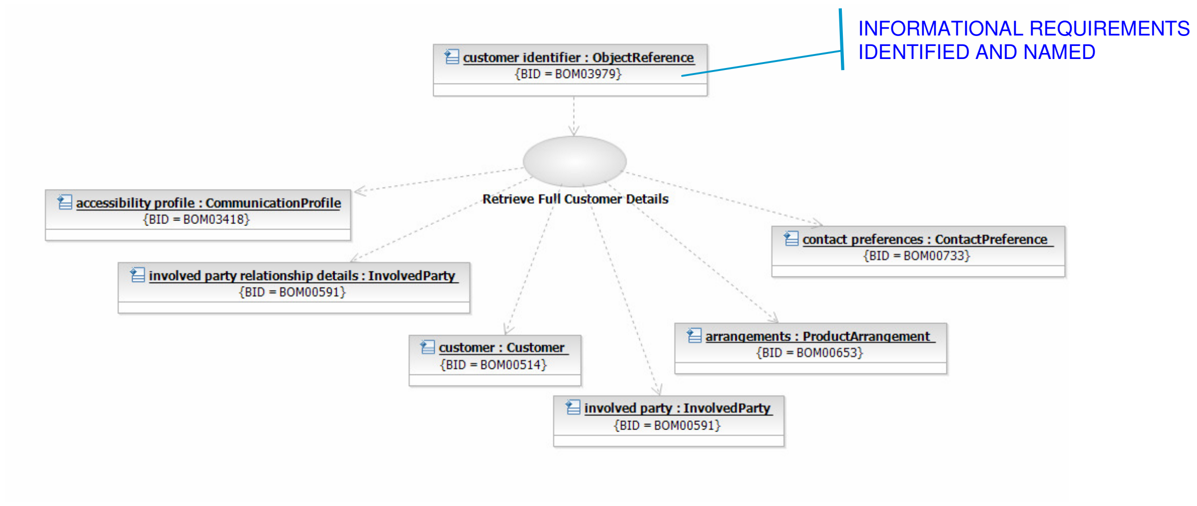

INFORMATIONAL REQUIREMENTS IDENTIFIED AND NAMED
customer identifier : ObjectReference
{BID = BOM03979}
Retrieve Full Customer Details
accessibility profile : CommunicationProfile
{BID = BOM03418}
contact preferences : ContactPreference
{BID = BOM00733}
involved party relationship details : InvolvedParty
{BID = BOM00591}
customer : Customer
{BID = BOM00514}
arrangements : ProductArrangement
{BID = BOM00653}
involved party : InvolvedParty
{BID = BOM00591}

Figure 33. Identifying informational requirements

This analysis builds upon the details captured in the business use case for the corresponding activity in the business process models, and the data flows in the processes in which that activity is reused. In the business use case and process model, initial analysis was done to capture the information that must be provided in order for the activity to be executed, and the details of the information that would be produced by that activity and returned to the caller. These details support the identification of the inputs and outputs of the use case respectively.

Note that while the informational requirements at the level of the process and the level of the use case are closely related, they are not identical. A process will contain contextual information that will not be present at the level of the underlying services. The normalization and structure of information will also frequently differ. Process data containers are often flat and unstructured presenting the information in a way that best suits the process at hand. By contrast the underlying services tend to require more structured and reusable oriented information.

3.7.5. Locating Types within BOM

These informational requirements are incomplete without some details of the types of each input or output. The next step is to identify a corresponding type for each informational requirement within the model. The class content of BOM consists of a set of business type definitions. In most cases a type will exist within BOM for the information requirement being expressed. Where a type cannot be identified within the models, a new class can be added.

Identifying the appropriate type in BOM to describe an informational requirement does require some knowledge of the contents of the models. Conveniently the class content of the models is also grouped into a number of packages to assist with this task.

Starting with the list of packages in BOM, for each package the analysis team should question whether the concept in question could belong to this package. This should lead naturally to a single or small set of likely candidates. Within those candidates, start at the root of the class hierarchy and work down through the subtypes until the required type can be found, or added. Working through the model content in a systematic way makes an apparently inconsumable amount of content navigable. Note that this BOM is the same model as the conceptual data model. This means that extensions to BOM based on use case analysis will be directly propagated to the data analysis domain. Similarly extensions to the model based on data analysis will be reflected into the UML domain and be available to use case analysts.

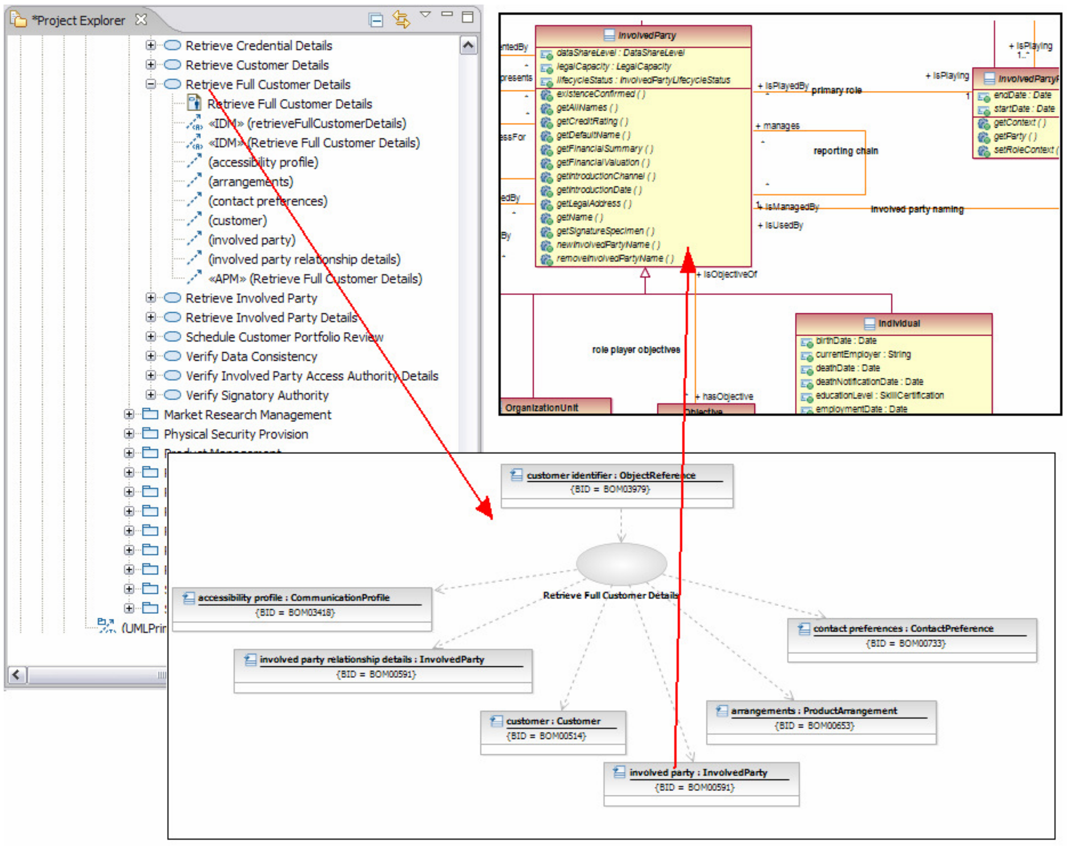

*Project Explorer
Retrieve Credential Details
Retrieve Customer Details
Retrieve Full Customer Details
Retrieve Full Customer Details
«IDM» (retrieveFullCustomerDetails)
«IDM» (Retrieve Full Customer Details)
(accessibility profile)
(arrangements)
(contact preferences)
(customer)
(involved party)
(involved party relationship details)
«APM» (Retrieve Full Customer Details)
Retrieve Involved Party
Retrieve Involved Party Details
Schedule Customer Portfolio Review
Verify Data Consistency
Verify Involved Party Access Authority Details
Verify Signatory Authority
Market Research Management
Physical Security Provision
InvolvedParty
Individual
OrganizationUnit
primary role
reporting chain
involved party naming
role player objectives
customer identifier : ObjectReference {BID = BOM03979}
Retrieve Full Customer Details
accessibility profile : CommunicationProfile {BID = BOM03418}
involved party relationship details : InvolvedParty {BID = BOM00591}
customer : Customer {BID = BOM00514}
contact preferences : ContactPreference {BID = BOM00733}
arrangements : ProductArrangement {BID = BOM00653}
involved party : InvolvedParty {BID = BOM00591}

Figure 34. Identifying types for information requirements

No customization of the details of BOM types is being made yet. Types are just being identified. The detailed attribution and behavior of those types will be analyzed later.

In many cases, there will be a requirement to use the type of a business concept as a use case input or output, and not the business concept itself. An example is the requirement to pass a selected Product Type in to a use case, as opposed to the details of the Product itself. Since typing in BOM is modeled using inheritance hierarchies, a convention of naming the input "product type" and defining its class as Product is used.

Note that this information analysis may be shared between the UML and ER domains. Many tools support the interchange of logical models between UML and ER notations, and the intent of the informational analysis in both domains is very much consistent. This means that changes to these models may be initiated though UML modeling and then propagated to the ER domain as a result of use case analysis. Equally, informational analysis occurring in the ER domain may be transferred to the UML domain. As such the class and ER models (particularly those supporting operational data) may be considered to be interchangeable under many circumstances.

3.7.6. Decomposing Use Cases

In many cases, this simple definition of a service candidate is not enough. Assuming a service is not a monolithic unit of code, we must account for and analyze potential delegations between service candidates to maximize the reuse, not just of services, but also the reuse patterns between services.

This requires that we look inside each use case candidate, and question the requirements surrounding their execution. To do this we must construct an Internal Use Case View. This involves providing an activity model below the existing use case to describe the execution scenarios (process models) that would be required to support a stated set of requirements. This activity model will consist of a set of finer grained activities, some of which will be reusable in themselves, and some of which will be simply local requirements.

Again, the relevance of this depends on the nature of the service being analyzed. Information services rarely require decomposition in this form. Decision services are often decomposed through structured rules analysis.

3.7.7. Identifying Nested Use Cases

Where these activities are reusable, they should be documented as separate use cases, so that the requirements they capture and realize can be reused in other use case decompositions. This will lead to the definition of the information requirements for these nested use cases, and so on.

The result of this is a model, not just of highly reusable service candidates, but also the relationships between these services in the form of delegation patterns.

3.7.8. Where Do I Stop?

Clearly this pattern could continue indefinitely, with use cases being decomposed into nested use cases, which are further decomposed. This raises the question of where should analysis stop?

During use case decomposition, there will come a point where that use case is no longer expressing a business requirement, but instead is becoming prescriptive as to the technology or solution that underpins that requirement.

This is particularly clear in the case of a retrieval service, for example to retrieve customer details. Decomposing this use case will lead to the definition of activities specifying that a connection to a data store is required, that an SQL statement should be constructed, etc. At this point the use case is no longer dealing in business requirements, but is attempting to dictate to a designer or developer, how those requirements should be satisfied. At this stage, use case decomposition has gone too far, and can safely stop for this use case.

3.8. Informational Analysis, Modeling and Planning

Section 3.7 above presents a method for the extension of the BOM based on requirements expressed through business use cases and business processes. It is important to recognize, however, that this is not the only source for informational requirements. Further, the core class model being extended here is inherently a normalized data model, which is very similar in nature to the models we expect following a pure data analysis/information engineering track. In fact, the core class structures in this UML may be interchanged directly with a conceptual data model expressed as an E/R model, driving consistency across the requirements being expressed through process and data analysis.

3.8.1. Create Develop Business Glossary

A business glossary, sometimes referred to as a data dictionary, defines the terms and data associated with a business area. In the industry models, this content is typically captured as part of the foundation models, and used to drive consistency of data definition across the model set. It is however common for different department or lines of business to have different definitions for the same, or similar terms. For this reason it is possible that a number of overlapping glossaries may exist within the enterprise. What is important is that they are used as a business entry point to consistent enterprise data architecture.

As an organization builds a glossary, it can include both semantics and representational definitions for glossary terms. The semantic definitions focus on precisely defining the meaning of the terms. Representational definitions include how terms are represented in an IT system such as an integer, string or date format. This means that business glossaries are one step towards creating precise semantic definitions for an organization. A glossary is the reference point for the meaning type and context of information that is used across the organization and will evolve during the SOMA identification, specification and realization phases.

An initial business glossary can be seeded from the Industry Foundation Models or from the BOM model directly. Since both models are tightly integrated, and contain the necessary depth of data definition, the choice of source can often be made on the basis of tooling capabilities. In either case the business glossary should be mapped directly to the underlying conceptual data model, which is in turn simply an alternative representation of the BOM.

In this way, data requirements expressed in either the UML environment as part of BOM, or in the E/R domain as part of the conceptual model are mapped in a consistent way to the business glossary, allowing a consistent data entry point to data requirements across the architecture.

3.8.2. Define Conceptual Data Model

A conceptual data model is a canonical data model at the conceptual level. A canonical model is one that is based on standardized or generic entities and can exist at the conceptual, logical and physical levels. This means that this model represents the strategic information requirements of the enterprise in a structured form.

This is exactly the same motivation as that of the BOM model above, and indeed using the Industry Models, the BOM model may be presented in the data analysis domain as a conceptual data model, and vice versa. This means that the definition of a conceptual data model is really the extension of an existing model (BOM) into in a conceptual data model context. It is through this interchange of a single model across modeling domains that we maintain the consistency that we require between the service consumer (business process) the service analysis (BOM) and data analysis (conceptual data model).

Entities are the fundamental building block of the conceptual data model. An entity represents a grouping of "attributes" around a single, key unifying information concept, and directly maps to the concept of a class in UML, stripped of its operational semantics. Each attribute represents a single information concept at the most granular level. Both attributes and entities are based on and aligned with the business glossary. In the conceptual data model, an entity will have "relationships" to other entities based on defined business relationships. These relationships also map directly to the relationships expressed in BOM.

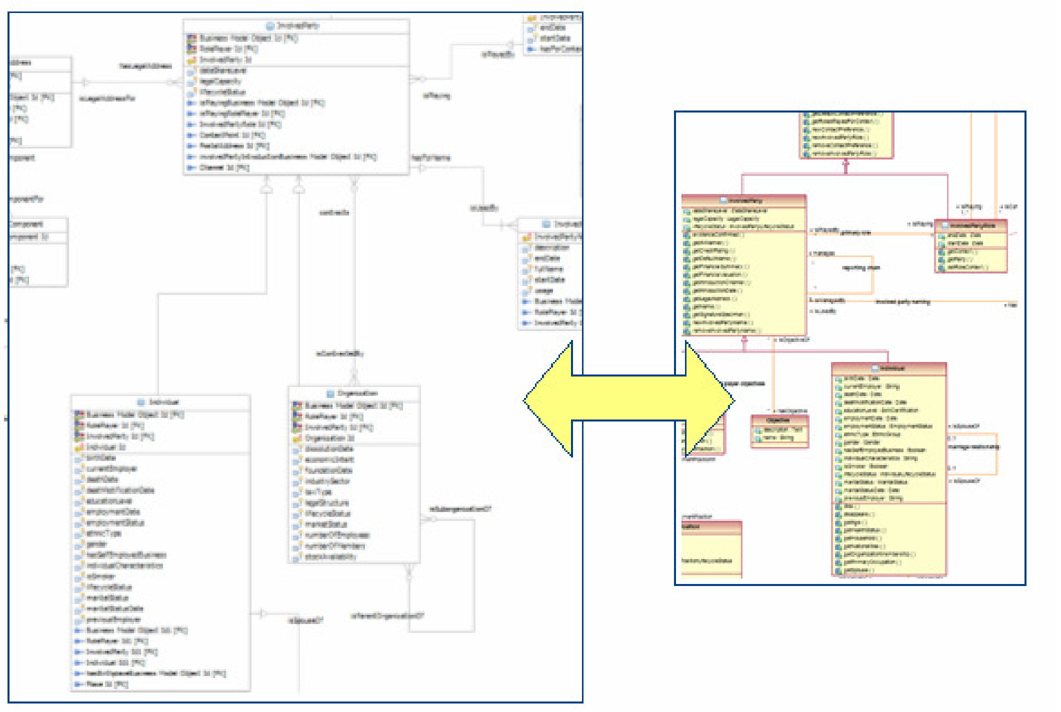


Figure 35. Model interchange between BOM and conceptual models

The conceptual model allows an understanding of the structure and content of the customers informational requirements. These information requirements equally support the analysis of service parameters in the UML domain, and data persistence structures within the E/R domain.

3.8.3. The Challenge of Reusable Types

An analysis level class model, such as those within BOM, or a conceptual model, simply expresses the details of a given business concept, and its relationship to other concepts.

The result of this analysis is a network of interconnected type definitions (such as classes) within which any one type is connected to many others. Pointing to a single type within this model, and trying to express it in isolation of the rest of the model, raises a question of where one type stops and the next starts.

This is precisely what is happening when the informational requirements of a use case are identified. Each information requirement expresses a type, which is backed up by the class models of BOM. But in the case of an example such as "Individual" or "Person", which of the connected types are also included? The name of the individual is likely to be required, but are the details of every Arrangement in which this Individual plays a role needed as well?

Traditional messaging approaches tend to define a data payload on a "per message" basis. This means that each message defines an entirely new type to describe its data payload. If type definitions are to be reusable, this approach can no longer apply.

Instead, a set of message independent type definitions are required that can be used to describe the parameters of each message.

This however implies a trade off. If type definitions are no longer tailored to a specific message, but are intended to be reusable across messages, then by definition types cannot be sensitive to their context or message.

Some amount of flexibility can be built in using optional type attributes, and by controlling the granularity of type, however it is a necessary compromise that for types to be reusable across business services, they must be defined as a “best common view” of the needs of those services.

At design time this problem will be addressed by manipulating relationships and interface definitions to define the actual scope of each type. This design level effort must be guided by requirements however, so it is essential to gather requirements pertaining to the boundaries of type within BOM.

3.8.4. Boundary of Type (BOT) Diagrams

Boundary of type diagrams are required where a type has a complex set of relationships to other types, and there is some ambiguity about which of those relationships is commonly required across all service contexts.

Note that this is not an exercise in exploring the mandatory elements of a type. This is captured through relationship cardinalities. This is an exercise in identifying what the most commonly used, and thus reusable boundary of a type is, across multiple services that use that type.

This does not preclude the use of additional details that are related to that type either. We are simply identifying that when we describe a given type, for example “Customer”, that certain information may also be passed as part of that aggregate. Additional details may be passed as further parameters of those services in which they are required.

If the boundary of type is described correctly, the resulting granularity of type definition will allow reasonably context sensitive groups of parameters that meet the needs of specific services. This provides a good compromise between the customization and low reuse of traditional messaging, versus the reuse driven, but non-customized forms of reusable type definitions.

To create a BOT diagram in BOM, a new class diagram called “<Type Name>BOT” should be inserted into an “Additional Views” section of the BOM package in question. For example “Customer BOT”. The type for which we are defining the view should be added to this diagram and its outgoing relationships to other types queried. For those that represent a relationship to a type that is always included in the master type’s details, then both the relationship and the target type should be added.

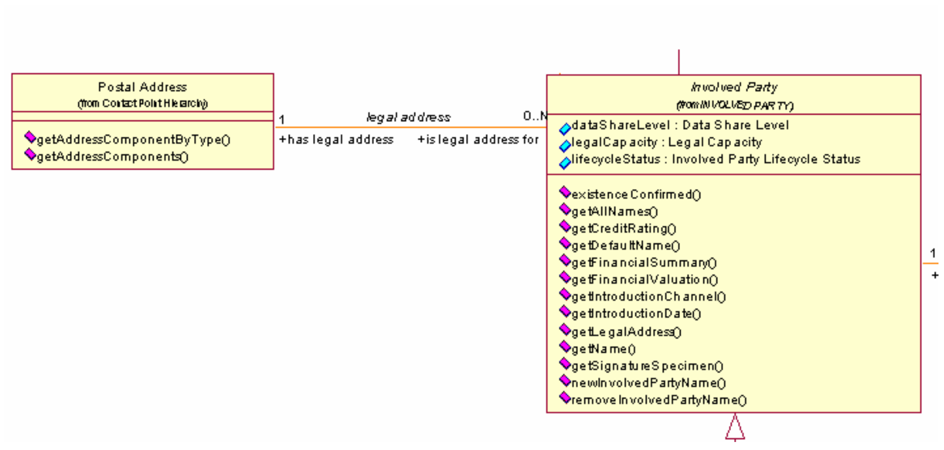


Figure 36. BOT diagram fragment - target class is required

An example of this is where a dependent type within the model, such as the legal address of an Involved Party, is considered always to be a valuable part of the Involved Party aggregate. In this case, the class Postal Address, is added to the common class diagram, along with the relationship that binds it to the Involved Party Class. This captures the information necessary to guide designers working in IDM in their customizations.

Where only a part of the target class is to be part of the common aggregate required, (for example, just the code from a Registration) this analysis can be captured by adding the target class to the diagram and annotating the parts of that target that are required.

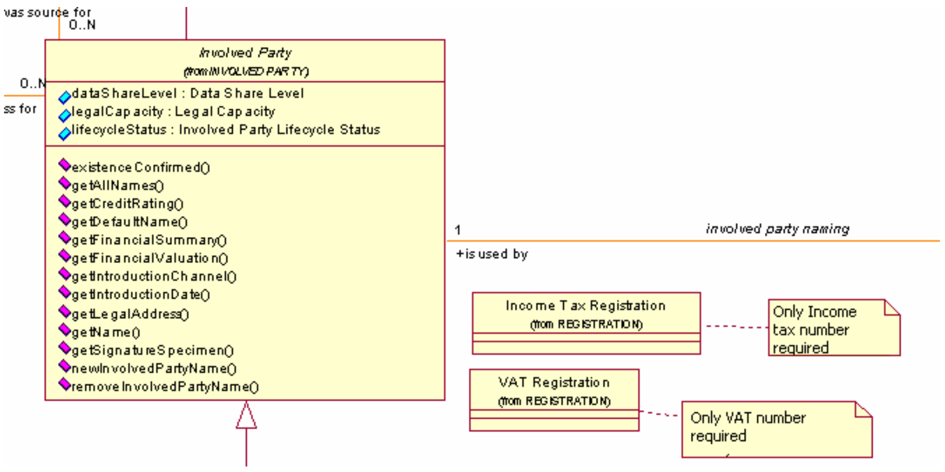


Figure 37. BOT diagram fragment - part of target class required

For example, a VAT registration number might be considered to be part of the common aggregate Involved Party, but the rest of the details of the VAT registration may not. In this case, the target class is added, without its complex navigation path and is annotated with a description of what specific aspects of that class are considered to be part of the common details of Involved Party.

This task of identifying the boundaries of type should be driven by use case analysis. Where a type is identified as the basis for an informational requirement, a BOT diagram may be needed for that type. If a diagram already exists, it can be validated against the new requirements under analysis.

Defining BOT diagrams may also alter the definition of inputs and outputs in use case external views. Analyzing the boundary of the "Customer" type may alter whether details of the "Customer Address" are passed as a separate parameter, or included within the "Customer" aggregate.

3.8.5. Capturing Data Requirements

Within UML class attributes are the simplest form of expressing the detailed data requirements for a type. An attribute with a specific type, and optionally an array specifier, captures simple static data requirements. All attributes within BOM should have a detailed business definition and a type specifier.

By convention, relationships between a class and another key business type (Business Model Object subtype, or dependent type) are not modeled as attributes. Instead, to allow a more detailed business definition and multiplicities to be expressed these are captured using UML associations.

A UML association may also have an Association Class, which provides addition data requirements that pertain to each instance of that UML association. Relationships between a type and primitive data types, enumerations or lifecycle statuses are expressed through class attribution.

A common area of customization of the BOM class model is through the extension of enumerated types. Enumerated types describe a known set of classifications of the type to which they are applied. For example "Gender" (Male, Female or Unknown) applied to the class "Individual" is a classification of that individual according to the gender of each individual.

When analyzing the data requirements it is these enumerations that are frequently added and extended. Each organization will have its own distinct dimensions of classification, and specific values within those dimensions, which will need to be added to the class model.

Sub-typing is another common method of extending BOM with additional requirements. Often, project requirements will provide an extra level of sub-typing below that specified in the standard BOM model. This results in organization specific representations of generic Industry Model constructs.

It is important that any extensions to the BOM model during the capturing of data requirements are made in accordance with the defined Industry Models Meta-model rules (These rules are described within the Process and Integration Models Meta-model document).

Note that these data requirements will be also be expressed through the conceptual data model, and enterprise data modeling. This discussion focuses on the use of the UML syntax for the expression of data requirements; however an E/R notation is equally valid. In practice both will exist, with the expression of these data requirements moving between the UML and E/R domains, being modified in both environments.

A common issue encountered during use case analysis is the availability of detailed data requirements. Process analysis and early use case identification will typically result in the identification of type level requirements; however identifying the details of these types can be extremely difficult. This interchange of content between the UML and E/R domains assists with this issue, allowing data requirements to be expressed from a variety of perspectives by a variant of stakeholders, yet be expressed in a single consistent model.

3.8.6. Adding Subtypes

Where a new business concept has been identified during analysis and that type is a further specialization of an existing type with BOM or the conceptual model, then the new concept can be inserted as a new subtype within the model.

In doing so it is important that the new type is in fact different than the base type within the model, and is not simply a customization of that type.

Sub-typing should be used where two types exhibit distinctly different behavior, but the subtype behavior is a specialization of the super type. Where only one type is actually required, sub-typing should not be used to add customizations to the model, the type in question should simply be customized directly.

3.8.7. Adding Other Types

Where a new business concept has been identified, but that type is not a specialization of any existing type, then an entirely new concept should be added to the model.

Where a new type is added, a detailed business definition that precisely identified the intent of that type must be supplied. This type then needs to be analyzed to describe its relationships to existing types, and those relationships then need to be entered as associations or relationships. Attributes and operations of that type should also be analyzed and added accordingly, to describe the data requirements and operational requirements of that type.

3.8.8. Adding Relationships to BOM

Where a new relationship is identified between two existing types in BOM, that relationship is added as a new UML association in the model. Note that BOM uses a principle of generalization of associations where possible. This means that associations that apply to many subtypes within the model may be expressed in a more general way at the level of a super type within the model, reducing the number of associations in the model as a whole. This is a valid analysis pattern where the targets of the relationship, and not the specific specialization of the relationship, are of particular business interest.

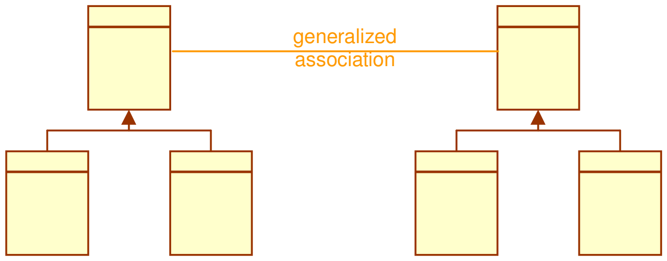


Figure 38. A generalized UML association

Where the specialization of the relationship is of key business interest, then the relationship should not be generalized, and a specific association should instead be added between the types in question.

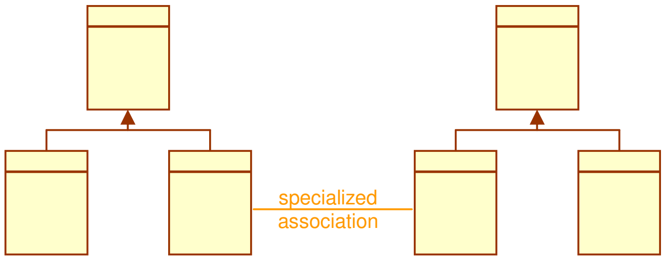


Figure 39. A specialized UML association

All associations added to BOM require a detailed business definition making the intent of that association clear. Associations also require a name, named roles (end points), and stated cardinalities. Aggregation and directionality of associations are not considered at analysis level.

3.8.9. Adding Attributes to BOM

Where analysis results in new details about the data attribution of a type, new class attributes should be added to existing classes within BOM. Where an attribute is added, a detailed definition of that attribute must be provided, as well as a type specification for that attribute. By convention, BOM only supports attributes of type enumeration, data type and lifecycle status. Where a data requirement, whose type is an independent or dependent type within the model, is identified, an association to that type should be used instead of a class attribute.

3.8.10. Adding Operations to BOM

Where analysis results in the identification of a new operational responsibility associated with a BOM type, that responsibility is modeled within BOM using an operation on the class.

Operations should be used to model responsibilities relating to a single type or its immediate dependent types. Where an operational requirement is identified that involves more than one business type, a use case is required to model the business activities involved in the interaction between types.

Operations in BOM are simply an identification of responsibility. No parameters or return types need to be identified. A detailed business definition is however required.

3.9. Variation Oriented Analysis

Variation oriented analysis is focused on the separation of the dynamic and static aspects of a business domain. This allows the identification of dynamic areas of the business and the externalization of their associated rules and structures, allowing future introduction of changes to the existing design. Variation oriented analysis may uncover additional service candidates that are part of anticipated future needs.

During variation oriented analysis, commonalities and variations are modeled through identification of variations in data patterns, business logic and business policies or rules. For example, the wide range of products offered by a financial institution involves a wide range of specific rules and data structures. However all these products still follow fundamental behavioral patterns and trends. Variation oriented analysis involves the identification of these patterns leading to the externalization of the rules and dynamic aspects of “Product”. This in turn has an effect on the business processes and services that deal with Product at both a generic and specific level.

Variation oriented analysis is a key step in the identification of business rules, and is strongly linked to the sections in this document that relate to the analysis and design of business rules, in particular sections 3.6 “Record Business Rules and Policies” and 3.6 “Rule and Policy Analysis”.

Externalized aspects of the business at analysis time can be captured through the use of specific subtypes and use cases, or the structuring of a rule based structure such as the IAA Product Specification Framework. In other cases variants may be captured through the use of external documentation such as the business use case documents described in this paper, and linked to the generalized service candidates identified within BOM, as described in section 3.6 “Record Business Rules and Policies”.

3.10. Rule and Policy Analysis

Having captured detailed textual requirements around business policy and business rule statements under section 3.6, the analyst will need to convert these into atomic business rules. A key requirement here is to express these rules in a structured way, while retaining an independence from the syntax of any target specific rules expression. There are a number of means of expressing rules content – many of them proprietary to a specific rules authoring environment. One means of expressing these rules in a standard way is through the use of SBVR.

3.10.1. About SBVR

Semantic of Business Vocabulary and Business Rules (SBVR) is a standard from the Object Management Group (OMG) that details a framework for the modeling of a business through its vocabulary and its business rules.

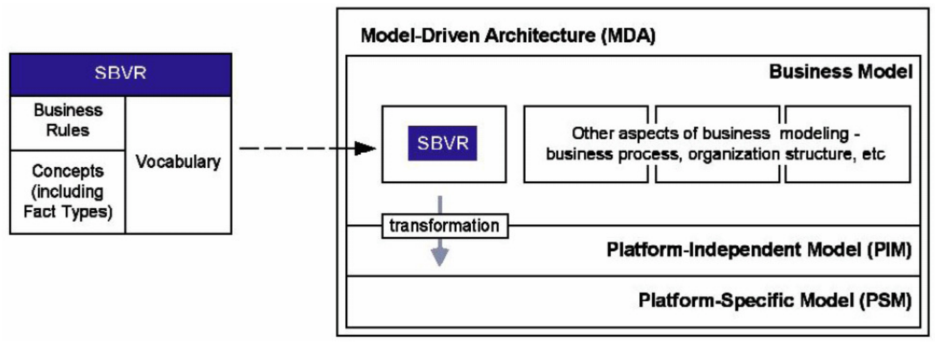

SBVR
Business Rules
Concepts (including Fact Types)
Vocabulary
Model-Driven Architecture (MDA)
Business Model
SBVR
Other aspects of business modeling - business process, organization structure, etc
transformation
Platform-Independent Model (PIM)
Platform-Specific Model (PSM)

Figure 40. Positioning of SBVR

This standard provides a detailed syntactical expression for atomic business rules that can subsequently be transformed into target specific rule formats, depending on the desired runtime expression of those rules.

The full specification of SBVR is extensive and available from the OMG – in this context we are simply concerned with its positioning as a means to capture structured rules analysis in a standard form.

SBVR expresses rules through the following type of elements:

term – as described earlier in this document, a term is an expression of a business concept, and is in itself the simplest form of business rule. Example: customer.

individual concept – a designation used for instances of terms. Note that a specific value of number such as *7* is an individual concept.

verb – a designation used for verbs prepositions or combinations of both. Example: a customer must ***provide*** identification.

keyword – a designation used for linguistic symbols that are used to construct statements. Examples: Each, the, a, exactly one.

Using these elements, it is possible to construct complex rules that capture the requirements as expressed by business rule statements in a structured and formal way. Additionally, these atomic rules may reference terms and relationships that are inherently part of a formalized model. Elements of type term may be derived directly from concepts expressed in the model – for example, customer, fixed term demand deposit or legal address. A term dictionary may be derived directly from the models, complete with business definitions, removing ambiguity from the definition of atomic business rules.

Similarly, the Industry models contains a formalized set of business verbs, which, coupled with formalized business terms, provide a precise definition of business behavior.

3.10.2. Analyzing atomic rules

We can see that our example business rule statement "…potential customers must be credit checked and identified..." as expressed by our business stakeholder in fact contains multiple discrete rules. We can also see that we can safely break this rule statement into distinct rules without losing any business information:

> "potential customers must be credit checked"
> "potential customers must be identified"

Further analysis may yield additional information as to the meaning of these elements. For example analysis of the lifecycle model of 'customer', could yield the following as per the Industry Models:

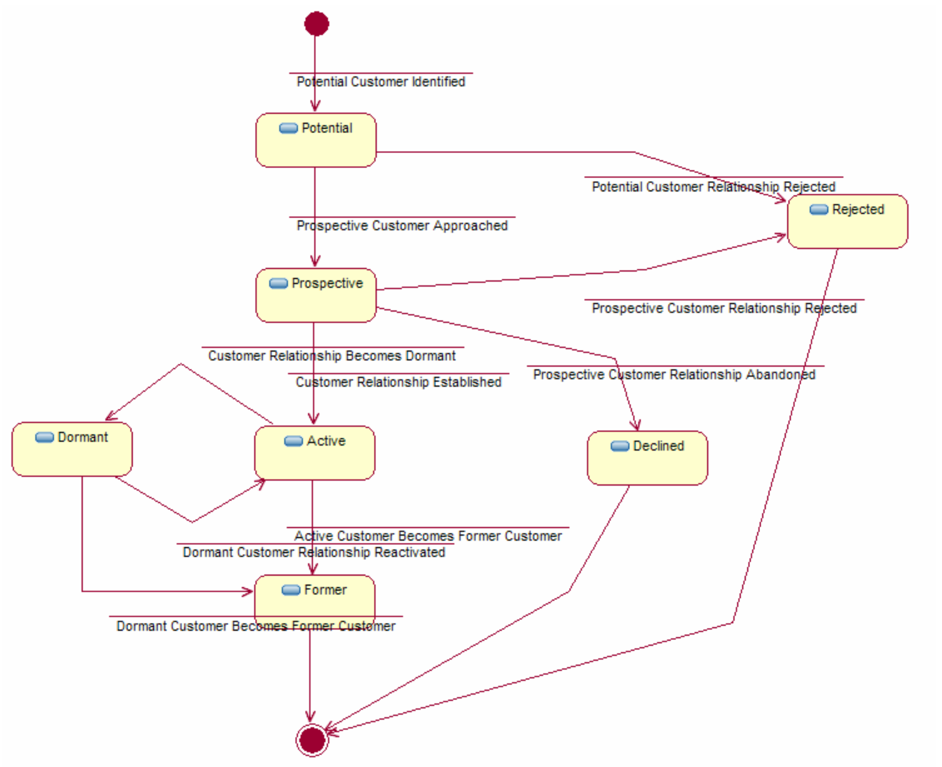

Potential Customer Identified
Potential
Potential Customer Relationship Rejected
Rejected
Prospective Customer Approached
Prospective
Prospective Customer Relationship Rejected
Customer Relationship Becomes Dormant
Customer Relationship Established
Prospective Customer Relationship Abandoned
Dormant
Active
Declined
Active Customer Becomes Former Customer
Dormant Customer Relationship Reactivated
Former
Dormant Customer Becomes Former Customer

Figure 41. State transition model for Customer

Based on this, we can see that 'potential' is in fact a lifecycle state for the concept of 'Customer'. In addition we now have critical information about the allowed states of the concept 'Customer', and the allowed transitions between states.

Similarly, further analysis into the meaning of 'identification' may yield the definition of further terms such as *'primary identification'* and *'secondary identification'*. These may be related directly to business concepts within the models, and relationships between those concepts.

Finally, analysis of related rules will often have an impact on atomic rule definition. For example, analysis of related contexts may highlight that identification of customers is critical, not just while they are in the potential state, but in all other states as well. This could in turn lead to a generalization of the rule – promoting it from the domain of "Potential Customer" to the domain of "Customer". However, for the purpose of this example, we limit our scope to the domain of "Potential Customer".

All of this analysis taken together would lead us to two atomic business rules:

> "customers in the potential state must be credit checked"
> "customers must provide primary and secondary identification"

Expressing these rules in SBVR would yield:

> "…each customer must *provide* exactly one primary identification and exactly one secondary identification …"

> "…a customer whose state *is potential* must *have* a credit check…"

3.10.3. Relating business rule statements to atomic rules

With the definition of increasingly formalized layers of business rules, traceability becomes a concern, not just to the models elements to which these rules relate, but between these levels of rules. As atomic business rules are defined they should be traced to the business rule statements from which they originate, which in turn relate to business policy. The nature of this traceability depends on the means used to express atomic rules.

3.11. Existing Asset Analysis

Existing Asset Analysis is the process of analyzing existing assets such as existing (legacy) systems, packaged or custom applications or industry standards in order to gather requirements and validate existing requirements against the capabilities of these assets.

It is critical that this bottom up analysis does not excessively bias the specification of business requirements. Using the analysis of existing assets to identify missed requirements will provide valid input into the analysis of business processes and services. However, the expression of these requirements must not be constrained or heavily influenced by those assets.

Existing asset analysis is a useful mechanism for identifying future issues in supporting proposed services at design time so that these constraints may be factored into project planning. Technical constraints related to existing systems should be evaluated as early as possible for risk management purposes; thus conducting technical feasibility of service realization decisions is often done as soon as possible after/during Existing Asset Analysis.

The final compromise from pure requirements to final solution is documented as part of the design models, with the analysis models capturing the pure requirements without technical limitations.

Chapter 4. Specification

As an input to this stage of model customization, all business analysis required to describe the service requirements has been performed and captured within BOM. This analysis can now be handed over to a design team that can extend the organizations' services blueprint in line with these requirements.

This team will be making design decisions with a component and interface design focus, to extend and define the component model of IDM, as well as making key decisions outside the bounds of the IDM model, for example the resolution of information services or the rules based implementation of decision services. SOMA Specification activities such as Service Specification, Subsystem Analysis, and Component Specification are accelerated by using Interface Design Model and Java Design Model from Industry Model as illustrated in the figure below.

Note that the steps in this phase are inherently iterative in nature. For example Service Specification often refines Subsystem analysis.

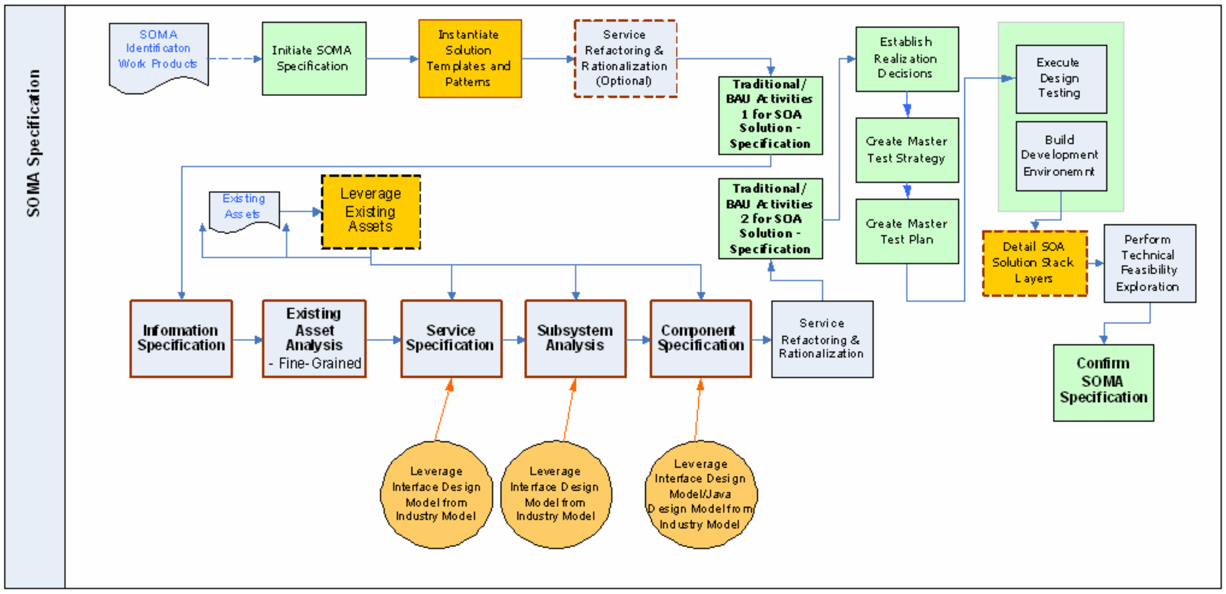

SOMA Specification
SOMA Identification Work Products
Initiate SOMA Specification
Instantiate Solution Templates and Patterns
Service Refactoring & Rationalization (Optional)
Traditional/ BAU Activities 1 for SOA Solution - Specification
Traditional/ BAU Activities 2 for SOA Solution - Specification
Establish Realization Decisions
Create Master Test Strategy
Create Master Test Plan
Execute Design Testing
Build Development Environment
Detail SOA Solution Stack Layers
Perform Technical Feasibility Exploration
Confirm SOMA Specification
Existing Assets
Leverage Existing Assets
Information Specification
Existing Asset Analysis - Fine-Grained
Service Specification
Subsystem Analysis
Component Specification
Service Refactoring & Rationalization
Leverage Interface Design Model from Industry Model
Leverage Interface Design Model from Industry Model
Leverage Interface Design Model/Java Design Model from Industry Model

Figure 42. Industry Models leveraged during SOMA Specification

4.1. Information Specification

At this point in the method an organization seeks to establish a detailed design of the information structures used to support services. Clearly, the definitions of these information structures must be related to the structures that are available within the data platform in order to have a chance of being realized. However, wholly adopting the structures exposed by the data platform directly may lead to poor service definitions and less reusability of services if these structures do not align to business requirements or the needs of service consumers. This introduces a 'balancing act' into service design – balancing the needs of service consumers against the reality of what the data platform can support, and identifying the most appropriate compromises to make in moving from service analysis to service design. It is also worth noting that there are a number of sets of information structures that may play a role here:

First there is the information structures defined within the information platform itself. These structures are optimized for persistence and data access and may be tightly bound to the implementation of existing systems. There may be multiple sets of these information definitions present due to the existence of multiple overlapping systems.

Next, this information platform is exposed through information services. Where possible the information structures exposed through these services should be structured in a way that is consistent with other categories of service such as decision services or process services. These structures should also align well to the needs of service consumers, rather than necessarily being a direct reflection of the underlying data platform. This means that the structures exposed as service parameters should be consistent across services, and is often an abstraction of the data structures used at the level of the data platform. The derivation of the structures exposed through services in IDM is largely driven by section 4.4 "Component Specification" below.

Finally service consumers in marshalling the parameters of the services they call will construct their own information structures. The nature of these information structures will depend to some degree on the environment being used to encode the service consumers and the scope and consistency of the services they consume, however it is desirable that there is a strong alignment between the information structures passed through services, and those used within the business logic of service consumers. Note however that these structures will never align completely. Service consumers use contextual information that is often of no interest to a service call. Similarly, highly reused services will often retrieve a small percentage of information that is of no interest to the specific context in question.

Formalizing these different representations of information and understanding the relationship between them key to the Information Specification of SOMA. Key to this phase is the identification of the various data representations in the data platform, and the mapping of those representations to the data structures of IDM. This can be performed at the level of the IDM model itself, or against a derived form of IDM such as generated XSD structures. Since the service definitions in IDM are initially well aligned to the needs of service consumers, these mappings will help to identify mismatches between the proposed data structures at the services layer and the capabilities of the underlying platform. These mismatches are then resolved by making modifications to the service designs in IDM to define a feasible service definition.

4.2. Subsystem Analysis

Subsystem analysis involves taking the functional areas of interest identified during service identification and translating these to formal subsystem definitions. These sub-systems are then decomposed to form functionally and technically oriented components. In many cases functionally oriented components (the Transactional components of IDM) will align directly to business functions.

4.2.1. Subsystem Dependencies

This activity of subsystem analysis is to formulate the dependencies and associations between subsystems. A subsystem that relies on a service from another subsystem is said to be dependent on that subsystem. This includes dependencies on data that may be accessed via services that provide the required data.

Subsystem and later component dependencies are key to the structure of IDM. A clear understanding of these dependencies is essential to form a well structured component model. This matter will be discussed in detail when structuring the components themselves and analyzing the interaction between services.

4.2.2. Functional and Technical Components

In decomposing a service component into its constituent functional and technical components, we have delegated the functionality provided by the service component to fulfill the subsystem's functional responsibilities. Functional components supply the business functionality required, while technical components provide generic functionality such as authentication, information access, error handling, auditing, logging, etc. Within the Industry Models, these component layers are Transactional and Utility components respectively.

Transactional Components
The transactional components of IDM are a layer of functionally decomposed components whose business definitions are derived from functional analysis. These components expose coarse grained and highly consumable business services, which meet the service requirements of business processes, client applications, and gateways. These services are derived directly from the top level use cases in BOM, and then collaborate amongst finer grained services to meet a particular business need.

Structural Components
The structural components of IDM are structurally oriented, and expose finer grained services than those of the transactional components.
These finer grained services are derived from elements within the use case decompositions of BOM, and thus support the services of the IDM business components, through a delegation pattern.

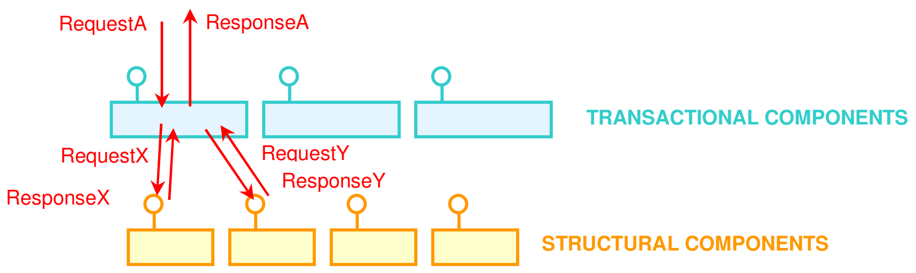


Figure 43. Delegation between Functional and Structural components

This delegation pattern, expressed as an interaction diagram in IDM, is directly derived from the use case decompositions of BOM, as highlighted in the following diagram:

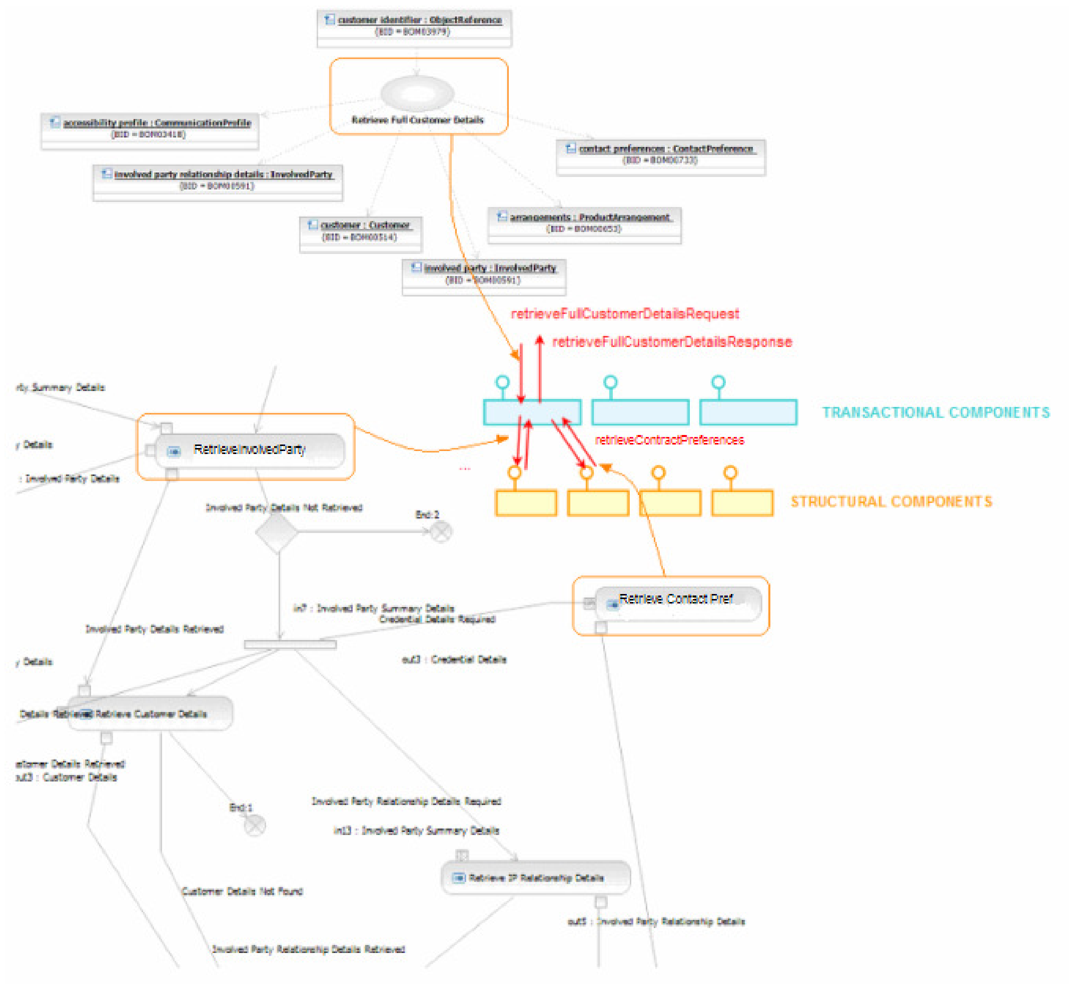

customer identifier : ObjectReference
(BID = BOM03979)
Retrieve Full Customer Details
accessibility profile : CommunicationProfile
(BID = BOM03418)
contact preferences : ContactPreference
(BID = BOM00733)
involved party relationship details : InvolvedParty
(BID = BOM00591)
customer : Customer
(BID = BOM00514)
arrangements : ProductArrangement
(BID = BOM00653)
involved party : InvolvedParty
(BID = BOM00591)
retrieveFullCustomerDetailsRequest
retrieveFullCustomerDetailsResponse
TRANSACTIONAL COMPONENTS
RetrieveInvolvedParty
retrieveContractPreferences
STRUCTURAL COMPONENTS
Involved Party Details Not Retrieved
End:2
Retrieve Contact Pref
in7 : Involved Party Summary Details
Credential Details Required
Involved Party Details Retrieved
out3 : Credential Details
Retrieve Customer Details
End:1
Involved Party Relationship Details Required
in13 : Involved Party Summary Details
Retrieve IP Relationship Details
Customer Details Not Found
out5 : Involved Party Relationship Details
Involved Party Relationship Details Retrieved

Figure 44. Relationship between BOM UCs and IDM services

Utility Components

The utility components of IDM provide platform specific services that are not directly related to the business content of the model, but are required indirectly to support it. Examples of utility components are Security and Logging.

4.3. Service Specification

This chapter is concerned with translating the use case content of BOM into extensions to the service architecture of IDM. This involves taking each use case within BOM and locating it within the component blueprint of IDM as a service. The informational requirements of this use case then drive the definitions of the parameters of that service.

Also of interest are the interactions between services. Analysis of these interactions has been captured within the use case decompositions of BOM, and can now be used to describe the design time collaborations between IDM services.
As discussed under section 3.5 “Identifying Service Patterns” there are a number of service patterns under consideration, each with subtle variances in their analysis / design needs. These variances can often become more pronounced during service design. Information services in particular can vary distinctly from process services. Process services tend to be strongly driven by business requirements, and become a collaboration of existing or new capabilities as design level. While process services do encounter constraints in the mapping to the capabilities of the environment in question, these constraints tend to influence the service specification in relatively subtle ways. By contrast, information services are often addressed by wrapping the capabilities of the information platform in a service and exposing it to the service consumer. This can often imply constraints around the data structures exposed as part of the information service. It is of course possible to enforce a mapping to reusable types that align with business requirements; however this can have significant performance implications. This means that the modeling of certain information services in IDM may be heavily influenced by the persistence layer, rather than the needs of service consumers.

4.3.1. Fine Grained Mapping to Existing Assets

The purpose of fine-grained mapping is to take the output from coarse-grained mapping performed during service analysis and determine the technical viability of existing applications and approaches to realize services.

Existing software assets and their dependencies and interfaces will have to be analyzed to determine if changes are required to support the business functionality. For example, in order to create a web services interface for a legacy implementation of a business function, analysis may involve the examination of the composition and flow of online transactions or batch jobs, or persistent data stores that help perform that function. The current design of these existing applications may have to change to support the functionality. There is also a need to identify any potential barriers to creating a web services interface with the desired quality of service. For example, a monolithic batch implementation of a business function may require sub second response time when invoked as a service.

In many cases this will imply a compromise on the part of the service definition at design time. The service that is ultimately designed and implemented may not align perfectly with the stated requirements due to technical limitations or other reasons.

4.3.2. Specify Service Operations

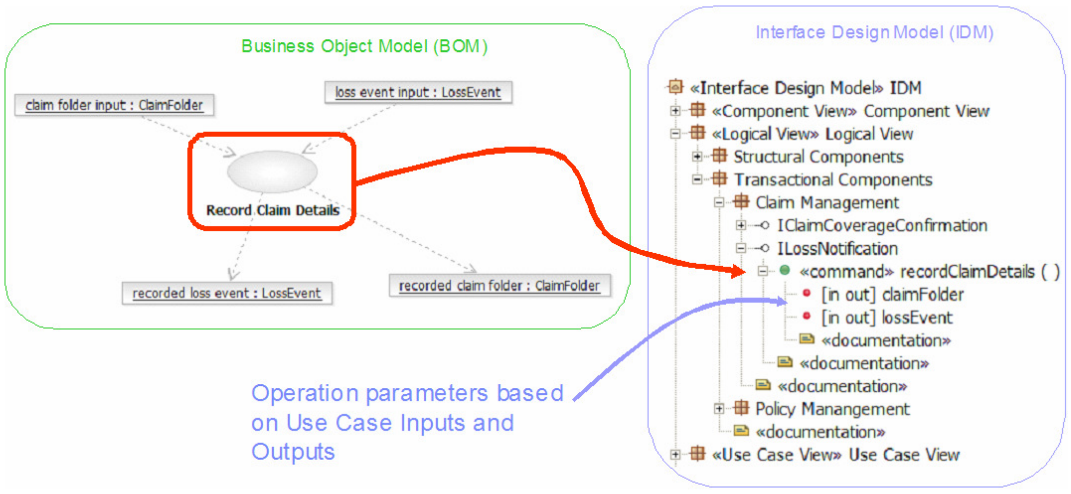

Business Object Model (BOM)
claim folder input : ClaimFolder
loss event input : LossEvent
Record Claim Details
recorded loss event : LossEvent
recorded claim folder : ClaimFolder
Interface Design Model (IDM)
«Interface Design Model» IDM
«Component View» Component View
«Logical View» Logical View
Structural Components
Transactional Components
Claim Management
IClaimCoverageConfirmation
ILossNotification
«command» recordClaimDetails ( )
[in out] claimFolder
[in out] lossEvent
«documentation»
«documentation»
«documentation»
Policy Manangement
«documentation»
«Use Case View» Use Case View
Operation parameters based on Use Case Inputs and Outputs

Figure 45. Service operations based on BOM use cases

Identifying Component Level

For each use case that has been added within BOM, that use case must be inserted within the component blueprint of IDM. This requires that the level of component be identified. Transactional components expose capabilities to the world outside the model. Structural components, while still visible to the outside world, are intended to support transactional component capabilities. So, for each use case the question that must be answered is whether this service is called directly from the “outside world” or whether it exists only to support another service.

Services that tend to be called from the outside world, such as “Perform Funds Transfer” can be pushed up the component stack into the transactional component layer. Components that are not called directly but support other services, such as “Get Source Account”, can be pushed down the component stack into the structural components.

It is a balance of these two forces, based on the perceived consumer of the service that determines what level of component any given service will reside. For this reason, an understanding of the context of a service candidate is very important. In the Industry Models this understanding of context is provided through the use of a top down analysis methodology based on business process analysis.

Identifying a Component

Having identified the level of component at which a service will reside, the next challenge is to locate the component that has responsibility for delivering this service. This involves the matching of a unit of work (a service) against a set of business functions (components) in order to determine which function has responsibility for performing this task.

IDM has a defined set of transactional components, so challenging each component in turn to determine whether this component has functional responsibility for the service in question is an easy way to address this. It is important to consider business definitions of the components, and not just their names during this exercise.

Identify an Interface
Each component in IDM exposes its capabilities through a set of interfaces. Each of these interfaces deals with a defined set of responsibilities, and it is important that the service candidate be assigned not just to the correct component, but also to the correct interface of that component.

Having identified a component that, based on its business definition, has functional responsibility for supporting the service in question, each interface of that component can also be challenged in question, based on its business definition, until a best match is found.

Extending the Component Blueprint
It is possible, that while trying to identify the correct component or interface for a service, that no existing business definition covers the service involved. In this case the component blueprint of IDM should be extended to include the new component or interface as required.

To preserve the consistency of the model, and to ensure that newly added components and interfaces do not overlap existing definitions, the results of earlier functional analysis should be consulted for an appropriate functional definition. This model hierarchically decomposes business functions and acts as the basis for IDM component and interface definitions.

It is also important to select the correct granularity of component or interface definition. The granularity of a given interface cannot be greater than that of the component that exposes it. Newly added components and interface should be of a similar granularity to their peers within the model.

Note that where a new component is added, the dependencies between that component and other components must be defined. These dependencies will be discovered though the addition of services to the component, and the analysis of the delegation patterns of that component.

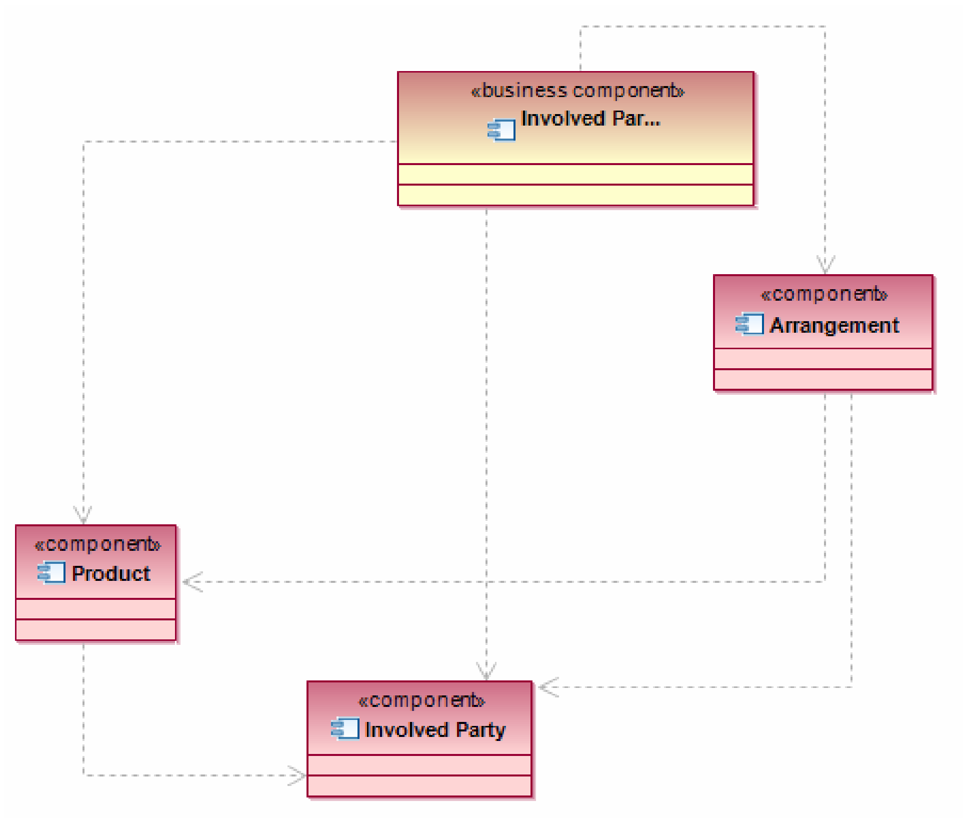


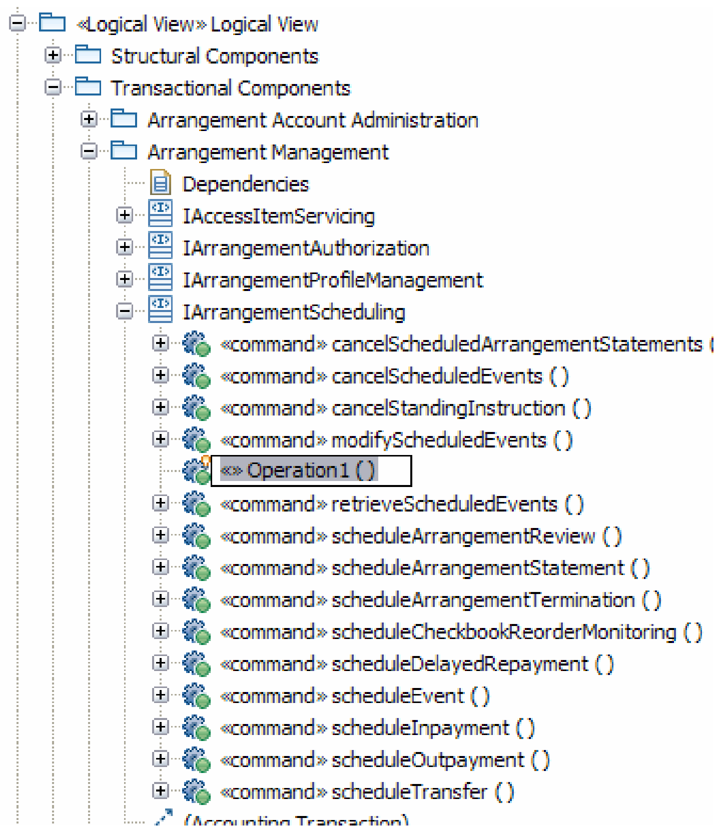
Figure 46. Component dependency diagram

Adding a Service Operation

Having located the component and interface within IDM that have responsibility for providing the service being considered, the service can be added as an operation of that interface.

Newly added operations should be added in a way consistent with the existing Industry Model meta-model, having a standardized return type and parameter specification mechanism.

The granularity of the service involved also affects the definition of the service through the use of the "<<command>>" stereotype. This stereotype is used to denote services that are called directly from client systems in a distributed environment. A useful way to think about this is to ask "would one send an XML message for this service". This will help to highlight the inherent difference between services such as "Add Involved Party" and "Set Date of Birth". For services that are likely to be useful in a distributed environment, the "<<command>>" stereotype should be added. Note that the delegation patterns and intent of IDM component layers will tend to result in "command" operations residing on business components, rather than structural components, although this is not an absolute.



«Logical View» Logical View
Structural Components
Transactional Components
Arrangement Account Administration
Arrangement Management
Dependencies
IAccessItemServicing
IArrangementAuthorization
IArrangementProfileManagement
IArrangementScheduling
«command» cancelScheduledArrangementStatements (
«command» cancelScheduledEvents ( )
«command» cancelStandingInstruction ( )
«command» modifyScheduledEvents ( )
«» Operation1 ( )
«command» retrieveScheduledEvents ( )
«command» scheduleArrangementReview ( )
«command» scheduleArrangementStatement ( )
«command» scheduleArrangementTermination ( )
«command» scheduleCheckbookReorderMonitoring ( )
«command» scheduleDelayedRepayment ( )
«command» scheduleEvent ( )
«command» scheduleInpayment ( )
«command» scheduleOutpayment ( )
«command» scheduleTransfer ( )
(Accounting Transaction)

Figure 47. Adding a service

Sometimes, the addition of a new business service will imply the addition of supporting or intermediate services. This is particularly the case where the business service requires access to data that is accessible through a generalized association, rather than a specific association. For example, a service “Retrieve Customer Details” might specify that the customer’s Market Segment forms part of the details to be retrieved. This implies a need to retrieve a market segment for a customer and also to set a market segment value for a customer.

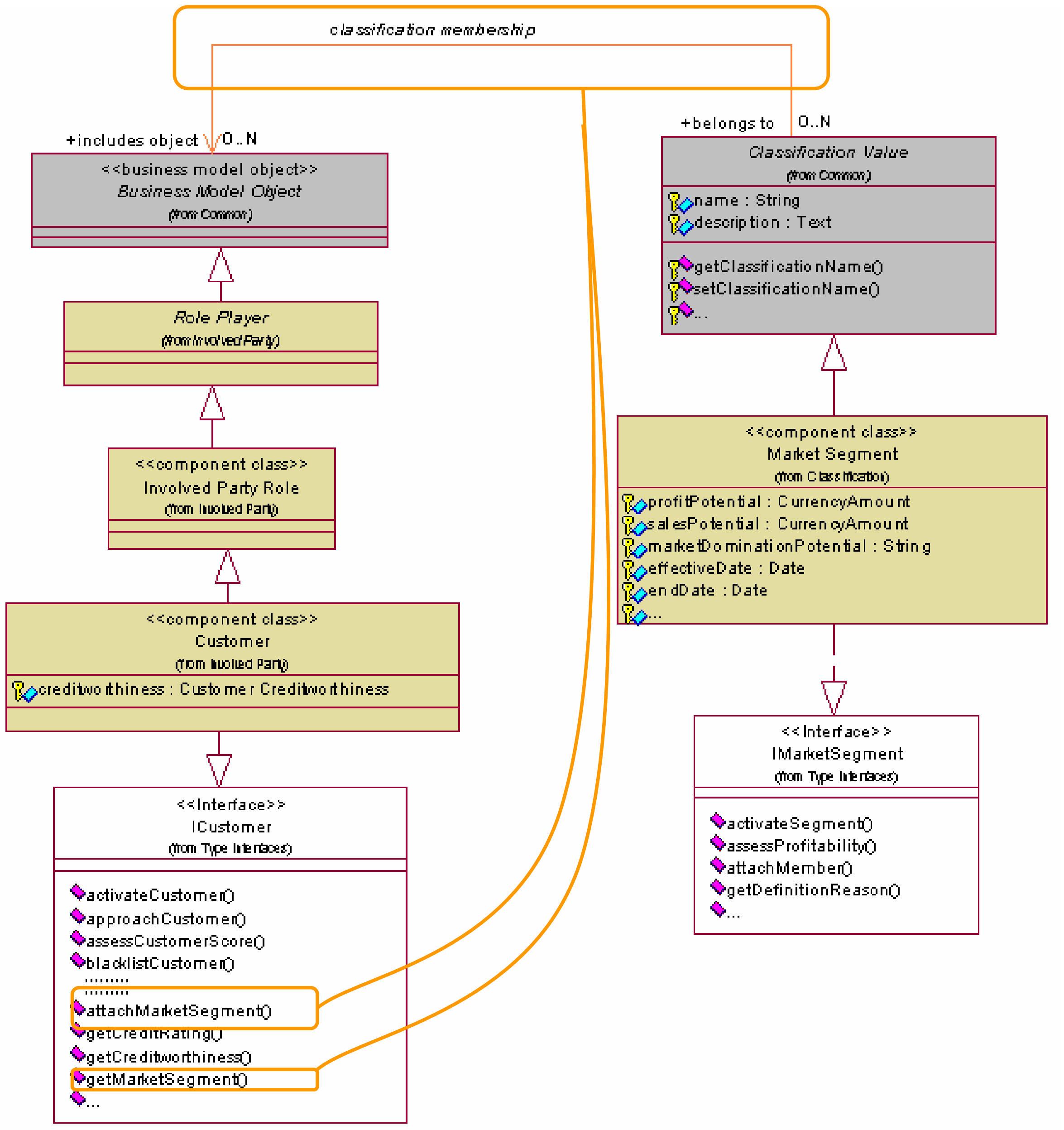


Figure 48. Operations traversing a generalized association

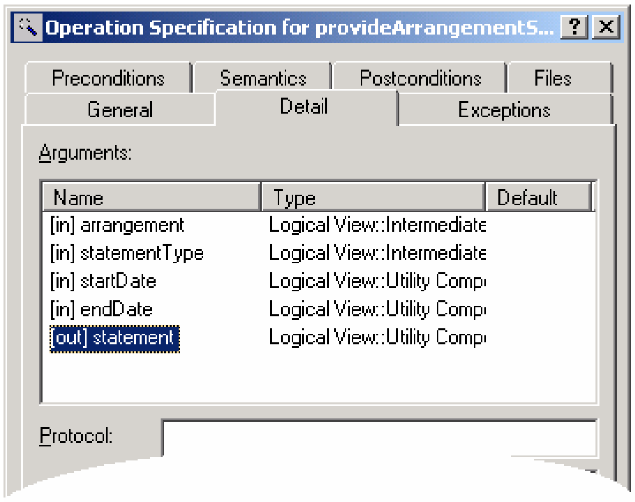
Following key steps are performed to leverage Interface Design Model during Service Message Specification:

- Update type definitions of IDM based on BOM customizations
- Define aggregations based on boundary of type class diagram in BOM
- Add stereotypes (attnav, typenav) to identify what is aggregated in a service message

Adding Parameters

Having located and established a business service within the component architecture of IDM, the service definition can now be extended to include the inbound and outbound data parameters of the service.

These parameters were captured during analysis (in BOM) as the input and output informational requirements of the use case that resulted in the definition of this service. For each input of this use case a new parameter is added to the service, with a name and definition, and a directionality specifier of "[in]". Similarly outbound parameters are added with a specifier of "[out]".
Where a given information requirement is passed in and then back out of a service, the qualifier "[in out]" is used.



Name	Type	Default
[in] arrangement	Logical View::Intermediate	
[in] statementType	Logical View::Intermediate	
[in] startDate	Logical View::Utility Comp	
[in] endDate	Logical View::Utility Comp	
[out] statement	Logical View::Utility Comp	

Figure 49. Adding parameters to a service

Identifying Parameter Types

Having added the required parameters for a service each parameter needs to be described using a type specifier. Types were analyzed for use case inputs and outputs within the BOM model, and this analysis can now be used to identify those types within the IDM model that meet the design level needs of this service.

Note that in the case of information services there may be limited freedom to express the parameter types of IDM in accordance with the requirements as stated within BOM. In these cases, the type structures expressed in IDM may be heavily biased by the underlying data platform. In extreme cases, this may invalidate the use of IDM at all for modeling information services that are effectively dictated by the underlying platform. However to overarching motivation to define a consistent services layer, with consistent type definitions should prevail where possible.

In many cases there is a direct mapping between classes in BOM and those within IDM. In other cases a class within BOM will map to an interface within IDM that exposes the capabilities of that class at design time. In these cases it is straightforward to identify the corresponding class or interface within IDM that provides the type definition required for a service parameter.

There are many cases however where a class in BOM has no obvious equivalent in IDM. For example where an extensive inheritance hierarchy at analysis time has been collapsed at design time, classes at the leaf levels of the inheritance hierarchy will not exist within IDM. They have been "rolled into" the definition of their super types.

Less frequently a requirement expressed at analysis time will result in a class at design time that is not directly stated in the analysis model. For example, typing (e.g.: "Product Type") is modeled in BOM through the use of inheritance hierarchies. In IDM this results in specific classes having an associated dynamic typing mechanism (e.g.: a "Product Type" class).

For these reasons maintaining a mapping between the class definitions of BOM and their equivalent structures in IDM is essential. In addition to this, maintaining a mechanism though which changes made to the BOM model can be readily identified and propagated to IDM will assist in driving model customizations through the model stack.

Any types added within BOM should be reviewed at design time for their impact on IDM. A mapping should be maintained of these customizations between BOM and IDM. The type specifiers for use case inputs and outputs in BOM can then be used to define the parameter types for services in IDM.

4.3.3. Model Service Dependencies

An effect of defining a service with specific parameters in IDM is that that component now has a dependency on the type definitions of its parameters. Specifically, this component must now have a dependency path through to the definition of each class or interface that is used to define its services. This will result in a customization of the dependencies of this component. This is one of several means in which dependencies may be established in the model. Other mechanisms include inheritance across component boundaries, and delegation patterns between services as part of an interaction diagram based on a use case in BOM.

When updating IDM, it is important to monitor the effect any update may have on the inter-component dependencies. This may determine to which component a particular class is assigned. A dependency between components exists in the following situations, detailed below.

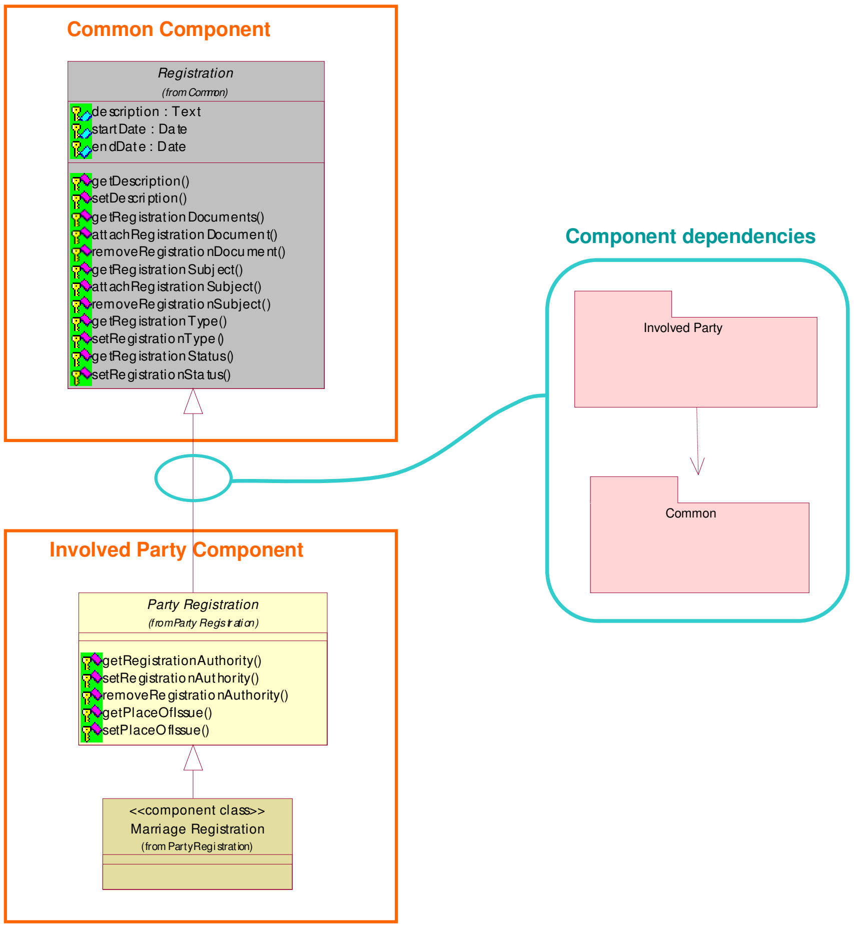
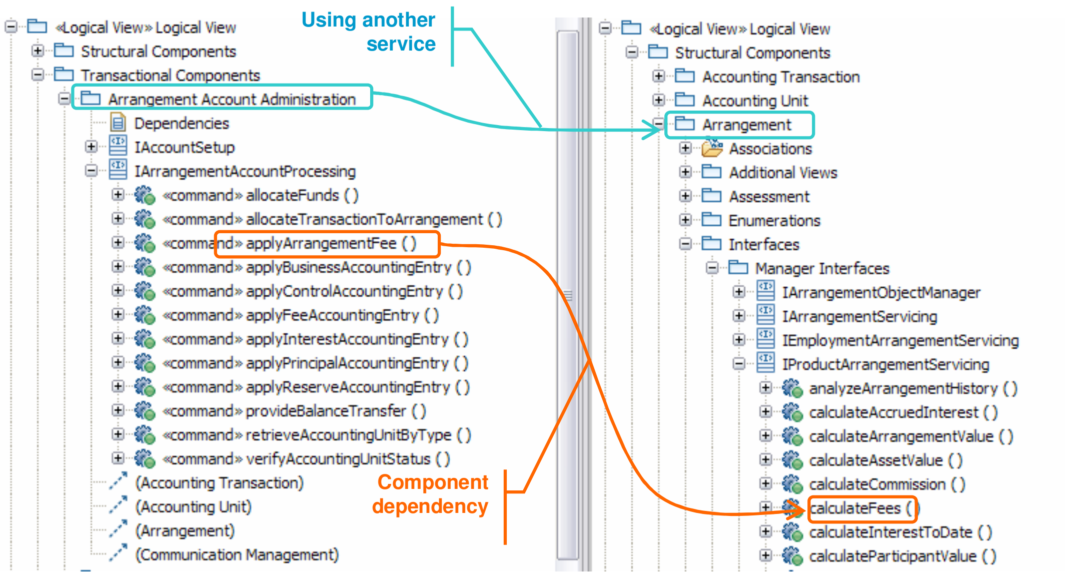


Figure 50. Inheritance across components

In this example, a type in one component inherits from a type assigned to a different component. This is a common inheritance pattern in IDM, particularly for general purpose business concepts such as Assessment, Classification and Registration. Where this pattern occurs, the component of the inheriting type must have a dependency on the component containing the super-type, either directly or through a dependency hierarchy. Thus, dependencies are identified based on the inheritance patterns in IDM.

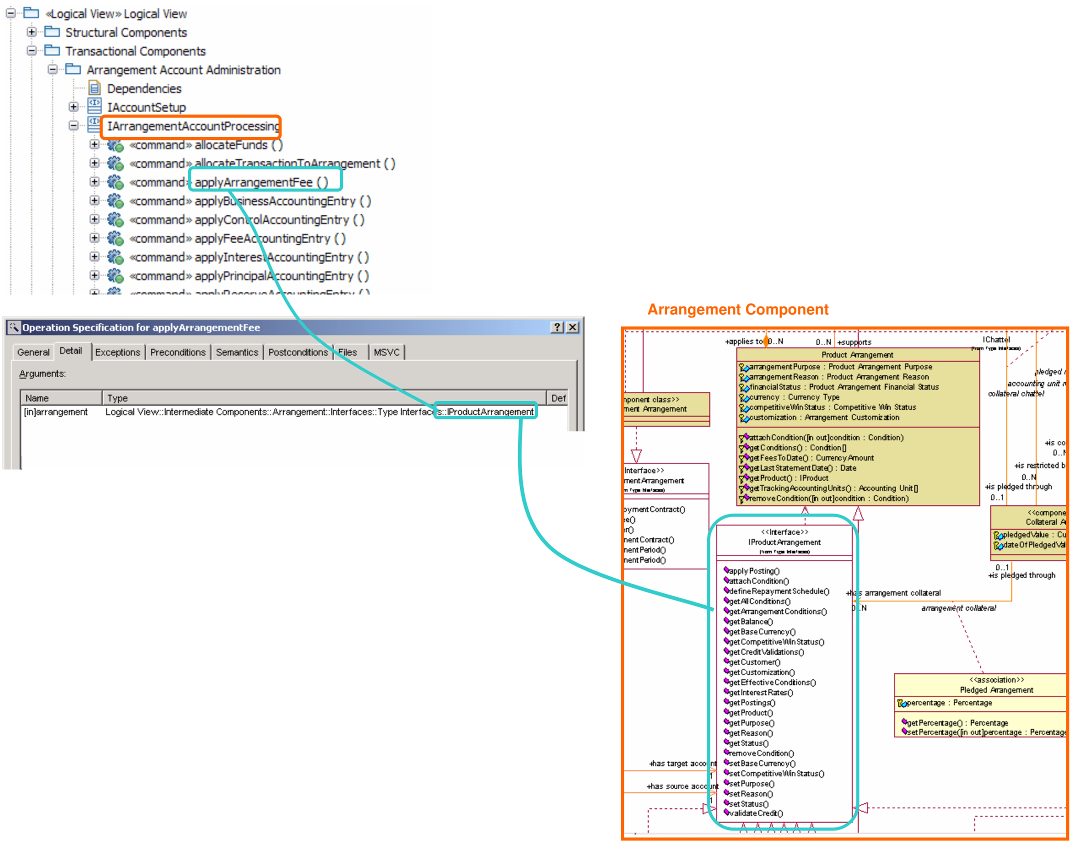


```
«Logical View» Logical View
  Structural Components
  Transactional Components
    Arrangement Account Administration
      Dependencies
      IAccountSetup
      IArrangementAccountProcessing
        «command» allocateFunds ()
        «command» allocateTransactionToArrangement ()
        «command» applyArrangementFee ()
        «command» applyBusinessAccountingEntry ()
        «command» applyControlAccountingEntry ()
        «command» applyFeeAccountingEntry ()
        «command» applyInterestAccountingEntry ()
        «command» applyPrincipalAccountingEntry ()
        «command» applyReserveAccountingEntry ()
        «command» provideBalanceTransfer ()
        «command» retrieveAccountingUnitByType ()
        «command» verifyAccountingUnitStatus ()
      (Accounting Transaction)
      (Accounting Unit)
      (Arrangement)
      (Communication Management)
```

```
«Logical View» Logical View
  Structural Components
    Accounting Transaction
    Accounting Unit
    Arrangement
      Associations
      Additional Views
      Assessment
      Enumerations
      Interfaces
        Manager Interfaces
          IArrangementObjectManager
          IArrangementServicing
          IEmploymentArrangementServicing
          IProductArrangementServicing
            analyzeArrangementHistory ()
            calculateAccruedInterest ()
            calculateArrangementValue ()
            calculateAssetValue ()
            calculateCommission ()
            calculateFees ()
            calculateInterestToDate ()
            calculateParticipantValue ()
```

Figure 51. Service dependency between components

A service in one component may use a service of another component. This creates a dependency from the component of the calling service to the component that provides the required service. These dependencies are identified through the collaboration patterns of IDM. In this example, the business component Arrangement Account Administration is dependent on the intermediate component Arrangement.



```
«Logical View» Logical View
  Structural Components
  Transactional Components
    Arrangement Account Administration
      Dependencies
      IAccountSetup
      IArrangementAccountProcessing
        «command» allocateFunds ()
        «command» allocateTransactionToArrangement ()
        «command» applyArrangementFee ()
        «command» applyBusinessAccountingEntry ()
        «command» applyControlAccountingEntry ()
        «command» applyFeeAccountingEntry ()
        «command» applyInterestAccountingEntry ()
        «command» applyPrincipalAccountingEntry ()
```

Operation Specification for applyArrangementFee

General | Detail | Exceptions | Preconditions | Semantics | Postconditions | Files | MSVC

Arguments:

Name	Type	Def
[in]arrangement	Logical View::Intermediate Components::Arrangement::Interfaces::Type Interfaces::IProductArrangement	

Figure 52. Service parameter type creating dependency

A parameter or return type of a service in one component may have a type that is defined in another component. In this example, the service “applyArrangementFee” uses the interface IProductArrangement in its parameter definition. IProductArrangement is defined in the Arrangement component. Thus the component Arrangement Account Administration has a dependency on the Arrangement component.

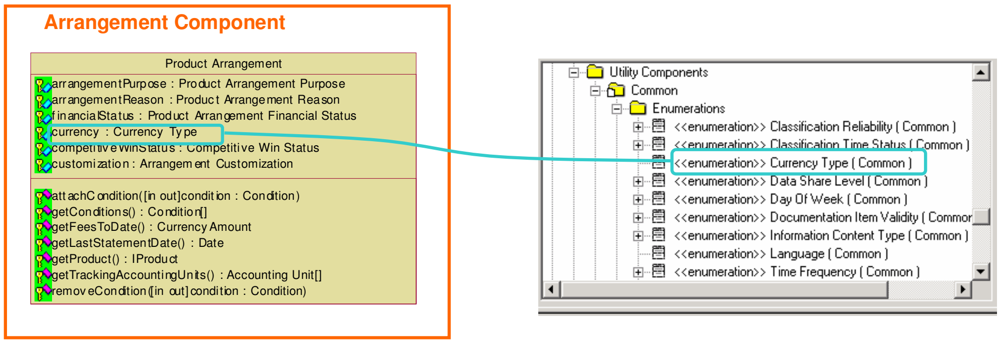

Arrangement Component
Product Arrangement
arrangementPurpose : Product Arrangement Purpose
arrangementReason : Product Arrangement Reason
financialStatus : Product Arrangement Financial Status
currency : Currency Type
competitiveWinStatus : Competitive Win Status
customization : Arrangement Customization
attachCondition([in out]condition : Condition)
getConditions() : Condition[]
getFeesToDate() : Currency Amount
getLastStatementDate() : Date
getProduct() : IProduct
getTrackingAccountingUnits() : Accounting Unit[]
removeCondition([in out]condition : Condition)
Utility Components
Common
Enumerations
<<enumeration>> Classification Reliability (Common)
<<enumeration>> Classification Time Status (Common)
<<enumeration>> Currency Type (Common)
<<enumeration>> Data Share Level (Common)
<<enumeration>> Day Of Week (Common)
<<enumeration>> Documentation Item Validity (Commor
<<enumeration>> Information Content Type (Common)
<<enumeration>> Language (Common)
<<enumeration>> Time Frequency (Common)

Figure 53. Attribute type creating dependency

Similarly, the definition of attributes for a type may use types defined in other components. In this example, the “currency” attribute of the Product Arrangement class has a type “Currency Type”. “Currency Type” is an enumeration defined in the Common component of IDM. Hence the Arrangement component has a dependency on the Common component.

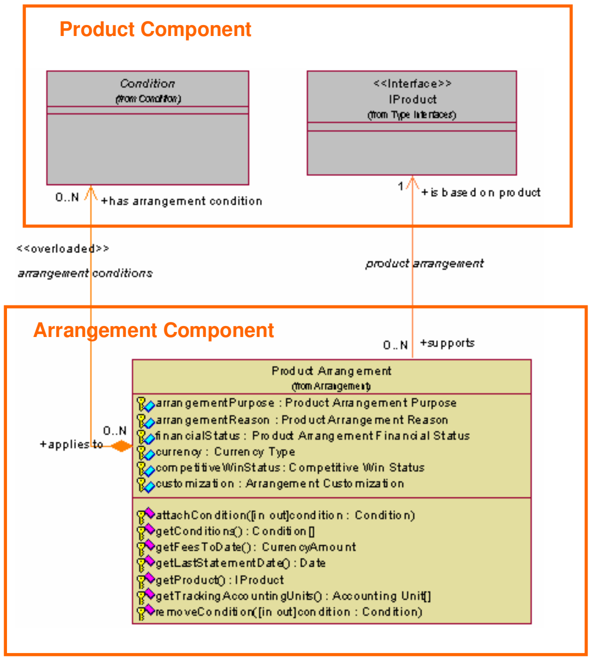

Product Component
Condition
(from Condition)
<<Interface>>
IProduct
(from Type Interfaces)
0..N
+has arrangement condition
1
+is based on product
<<overloaded>>
arrangement conditions
product arrangement
Arrangement Component
0..N
+supports
+applies to
0..N
Product Arrangement
(from Arrangement)
arrangementPurpose : Product Arrangement Purpose
arrangementReason : Product Arrangement Reason
financialStatus : Product Arrangement Financial Status
currency : Currency Type
competitiveWinStatus : Competitive Win Status
customization : Arrangement Customization
attachCondition([in out]condition : Condition)
getConditions() : Condition[]
getFeesToDate() : CurrencyAmount
getLastStatementDate() : Date
getProduct() : IProduct
getTrackingAccountingUnits() : Accounting Unit[]
removeCondition([in out]condition : Condition)

Figure 54. Associations across component boundaries

Associations frequently cross component boundaries as in the example above. This creates a dependency from the component containing the starting point of the association to the component containing the association target. These dependencies can be identified by reviewing the associations in IDM. In this example, the Arrangement component has a dependency on the Product Component.

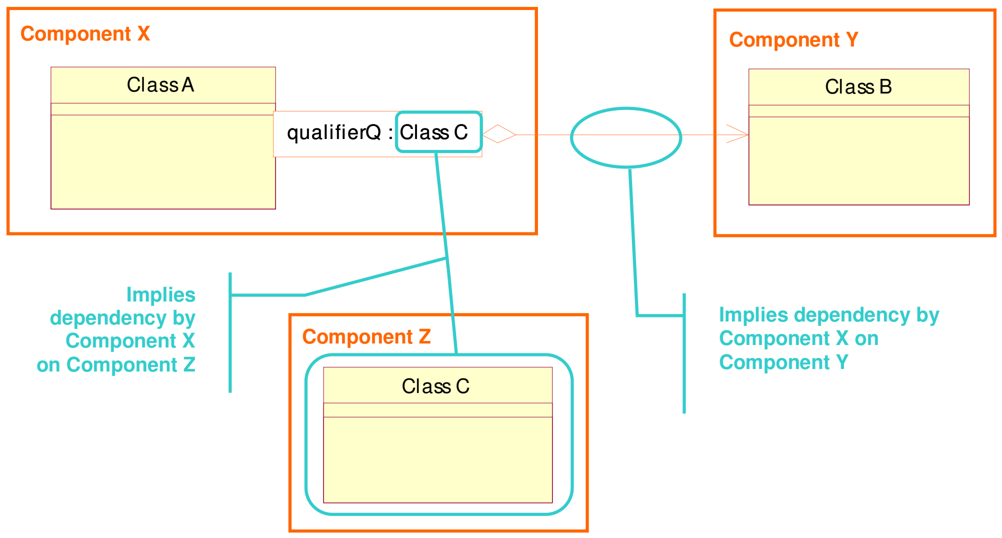

Component X
Class A
qualifierQ : Class C
Component Y
Class B
Implies dependency by Component X on Component Z
Component Z
Class C
Implies dependency by Component X on Component Y

Figure 55. Qualified association between components

Where an association is qualified, the type of the qualifier must also be taken into account. If the qualifier type resides in a component other than the components containing the source and target of the association, this will result in an additional component dependency. The component of the source type will be dependent on the component containing the qualifier's type definition. In the example above, the association implies that Component X is dependent on Component Y. In addition, the qualifier uses ClassC, defined in Component Z. Therefore, Component X is also dependent on Component Z.

4.3.4. Service Composition and Flow

To complete the customization of IDM, and to complete the design of business services, interaction diagrams need to be defined, where required, to illustrate how the services documented within the model collaborate to support a specific business need. Note that this step often results in modification of service dependencies, as per the preceding section.

These interaction diagrams are built upon the analysis of the internal use views of BOM. These views show the analysis of the internal workings of a service candidate – what business steps must occur in order to support that service candidate.

IDM interactions are used to define the internal workings of a business service. An interaction diagram shows a single scenario of execution within a service, and is required where the implementation of a service is of business interest and to clarify the delegation pattern to other services.

Typically, where a use case has an internal view in BOM describing the steps involved in executing that use case, then the corresponding service in IDM will have one or more interaction diagram.

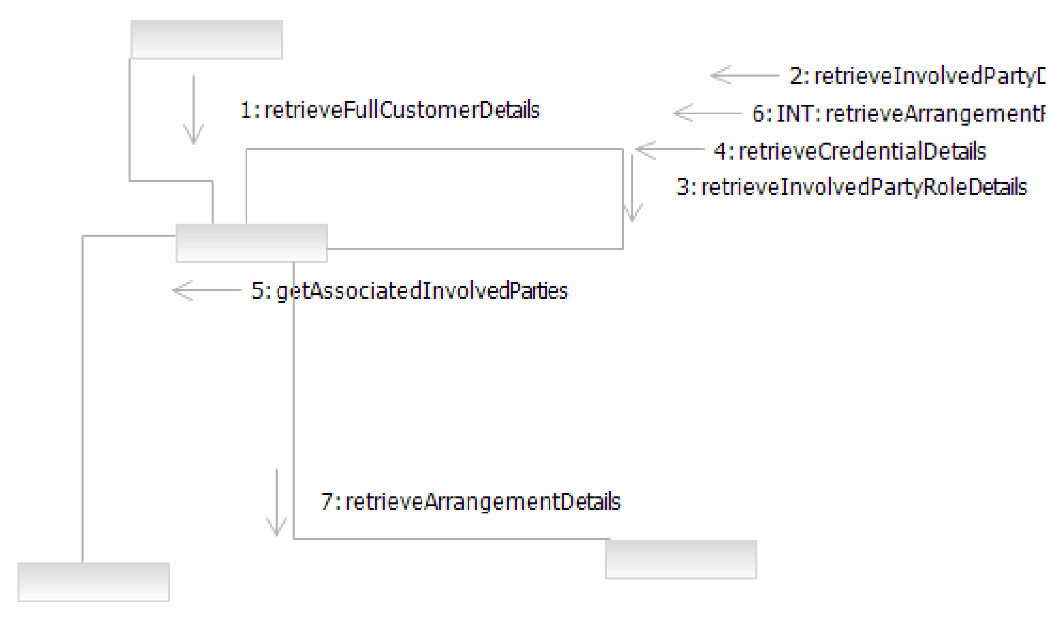

1: retrieveFullCustomerDetails
2: retrieveInvolvedPartyD
6: INT: retrieveArrangementI
4: retrieveCredentialDetails
3: retrieveInvolvedPartyRoleDetails
5: getAssociatedInvolvedParties
7: retrieveArrangementDetails

Figure 56. Sample interaction diagram

Since interaction diagrams only show a single execution scenario, a use case that can execute in a number of different ways will require a number of different interaction diagrams. At a minimum the primary flow of the use case should be illustrated as an interaction diagram.

By convention, interaction diagrams are named similarly to the service that they document. Where multiple interactions exist for a single service, then each should be named as per the service, with a qualifier in parentheses identifying which scenario the interaction diagram represents. For example; “Implement Rollover (Outpayment)”.

IDM interactions are represented within the component that houses the service that they describe. Mappings are also maintained between a use case and the interactions that result from that use case at design time.

A use case internal view in BOM represents a sequence of steps that occur on executing that use case. Not all of these steps will be candidates for reusable services. Many of these steps will simply be local logic that acts as the glue between service invocations.

These steps cannot be discarded at design time as they are essential to the requirement being analyzed, however they will not be supported by service candidates in IDM.

These steps become internal service logic, and are modeled in IDM within interaction diagrams by operations with the prefix “INT:”.

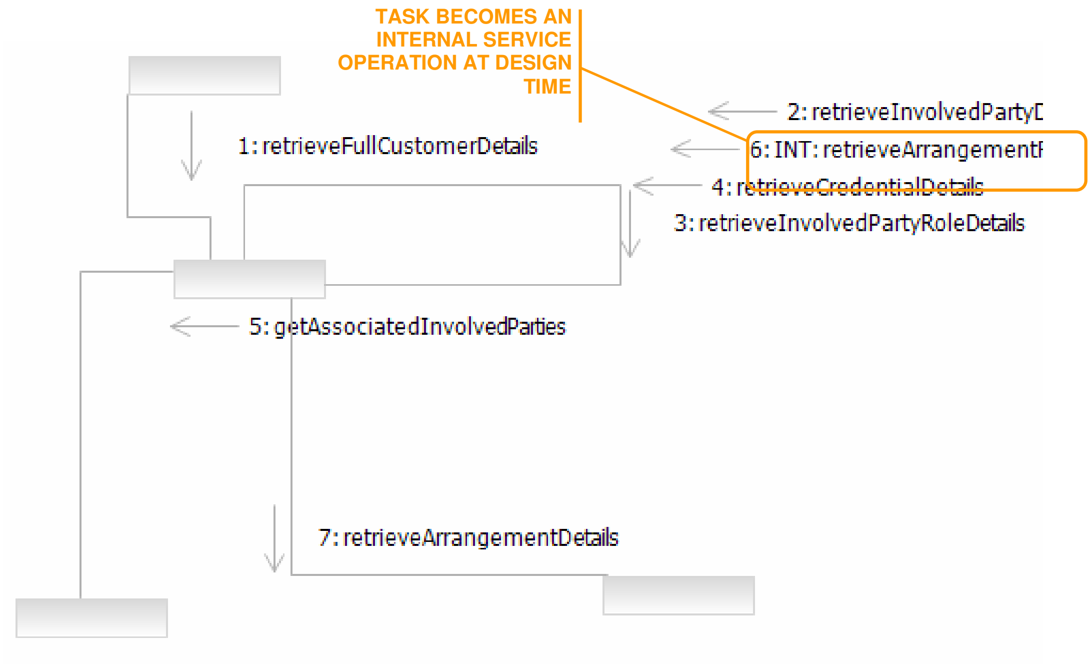

TASK BECOMES AN INTERNAL SERVICE OPERATION AT DESIGN TIME
1: retrieveFullCustomerDetails
2: retrieveInvolvedPartyD
6: INT: retrieveArrangementI
4: retrieveCredentialDetails
3: retrieveInvolvedPartyRoleDetails
5: getAssociatedInvolvedParties
7: retrieveArrangementDetails

Figure 57. Internal logic in an IDM interaction

The construction of interaction diagrams will have an effect on component dependencies within the model. Identifying that the implementation of a service invokes a number of other services implies a dependency between the components on which those services reside. Each component invoked within an interaction must be a dependent of the source component, and component dependencies must again be validated to ensure that linear component architecture is maintained.

4.3.5. Associate Business Rules and Policies

Section 3.10 “Rule and Policy Analysis” established a catalog of atomic rules based on business rule statements and policies. These rules must now be transformed into platform specific formats as rule expressions. A rule expression may take on a wide range of forms, and a key design decision is the identification of which rules are best expressed in which form. Externalizing business rules comes with an additional design time and run time cost. Which rules does it make sense to push out to a rules engine, and which can safely be hard coded?

Typical candidates for externalized rules are those which experience a high degree of flux in requirements. The primary benefit of using a rules engine over ordinary coding techniques is that rules engines allow efficient and controlled change of business rules as requirements change. Rules governing product behavior, terms or conditions often change based on market needs and should be externalized for flexibility.

Complexity of rules is also a factor. Most rules environments have useful capabilities to manage compound rules, decision tables and interdependent rule elements. Rules with a high degree of complexity, such as organizational policies which although they may experience less change in requirements are good candidates for rules based implementations because they are often difficult to express in ordinary code.

A third factor is reuse. As we have seen, rules are often nested in nature. And fragments of rule may be highly reused. While certainly attainable in code, building these fragments within a rules environment allows better control and management of reuse, particularly where rules are dynamic or complex.

Exposing business rules as services eliminates much of the complexity associated with the variations in rule expression syntax and rules engine specific support. Identifying a reusable service definition to encapsulate atomic rules allows that rule to be exposed as a web service, expressed in WSDL, isolating the client of the rule from the complexities of how that rule is implemented. For this reason it is important to maintain a strong connection between atomic rules and the parts of the business model in which they originated. Many business rules may be mapped to business activities, which in turn drive the definition of business services.

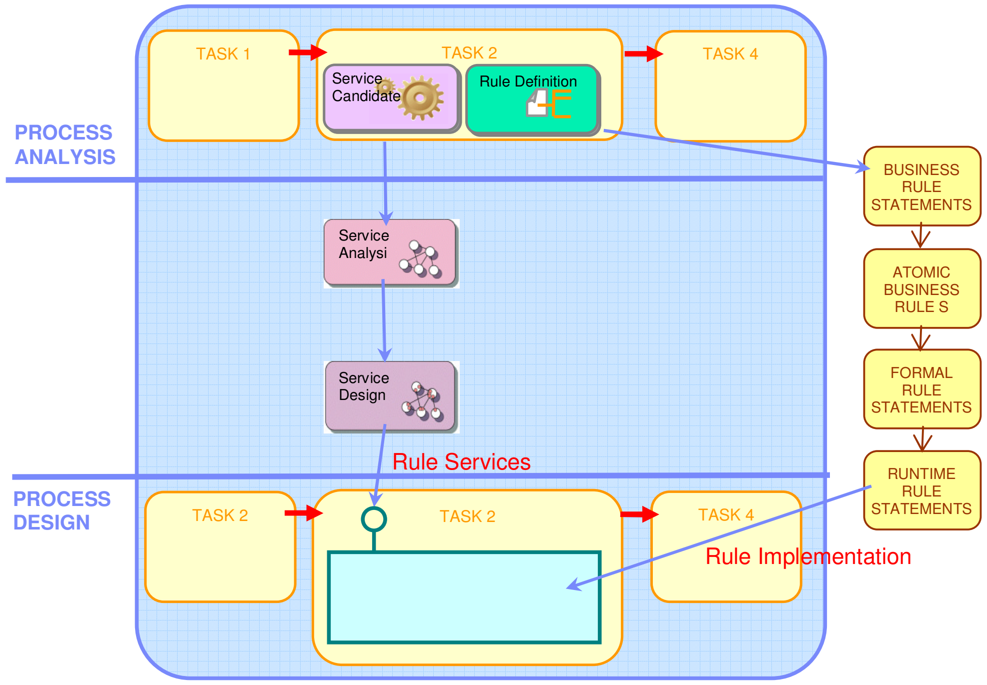


Figure 58. Defining rule services

Retaining this link provides a means to analyze and define the services that encapsulate rules in a way that is independent of the rule definition itself. Additionally, rules can then operate on reusable business objects, derived from IDM, rather than allowing each rule to operate on its own specific data sets and platform specific representations.

Having established a consistent set of WSDL interfaces through which rules may be invoked; those rules may be implemented using a wide range of technologies, without forcing the clients of those rules to accommodate a range of different techniques.

4.4. Component Specification

Component specification involves detailing the service components and associated technical and functional components in terms of the events the components must sense and respond, data attributes, rule and policies, interface definition and input and output messages, and internal flow.

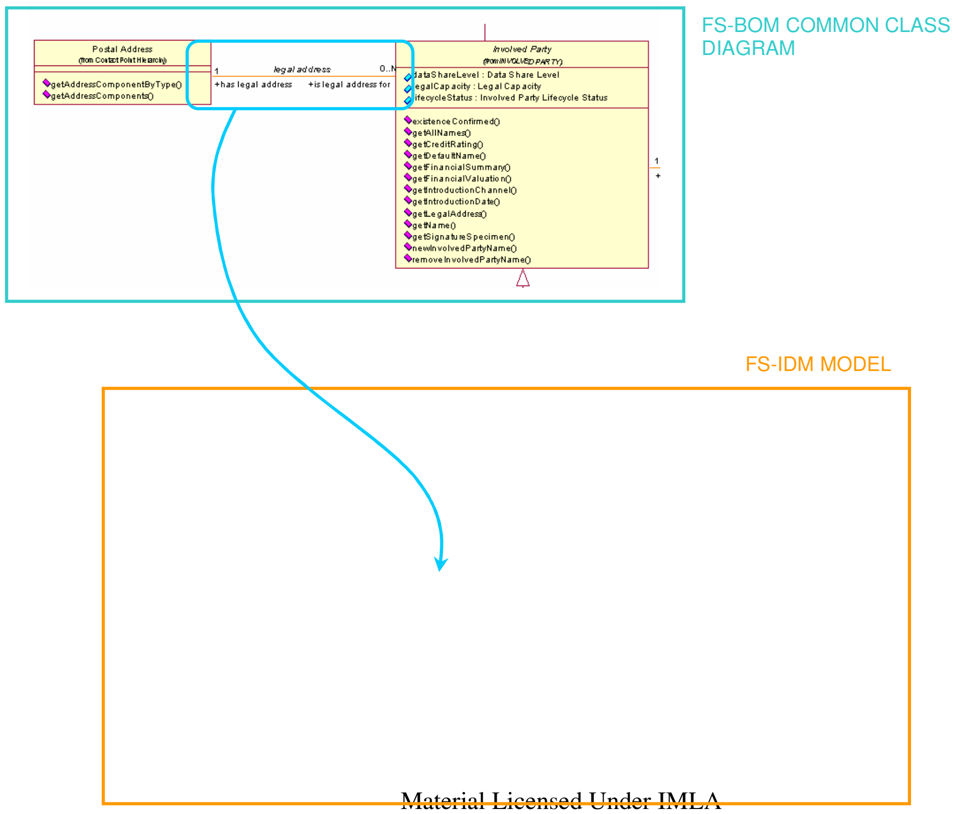
4.4.1. Applying OOAD

The activities of SOMA and OOAD are complementary and performed iteratively. OOAD should be applied during service specification and subsystem analysis in an iterative manner.

During analysis, a wide range of customizations will have been made to BOM that will affect the class definitions of the model. These customizations represent the detailed business requirements of the project and the organization in general, and must now be propagated to IDM.

These customizations include changes to the attributes, operations and UML associations of the model, as well as analysis of the boundary of type, expressed as Common Class Diagrams.

Interpreting Boundary of Type
The Boundary of Type (BOT) diagrams in BOM capture requirements surrounding the logical boundaries of key types within the model. This analysis is used to drive the definition of aggregated relationships and <<attnav>> (derived attribute) operations within the design model.



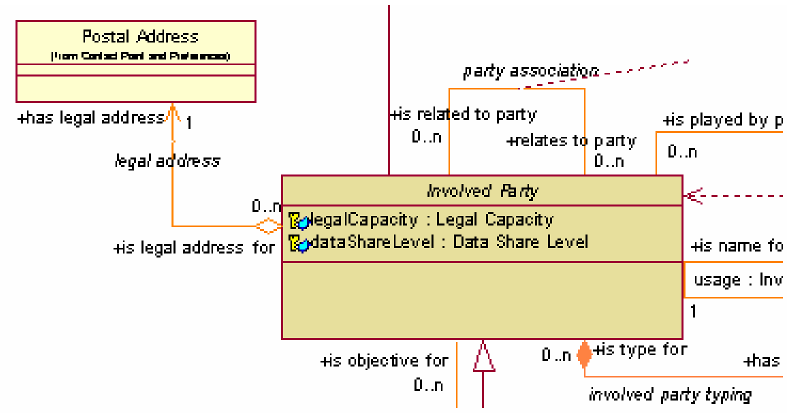
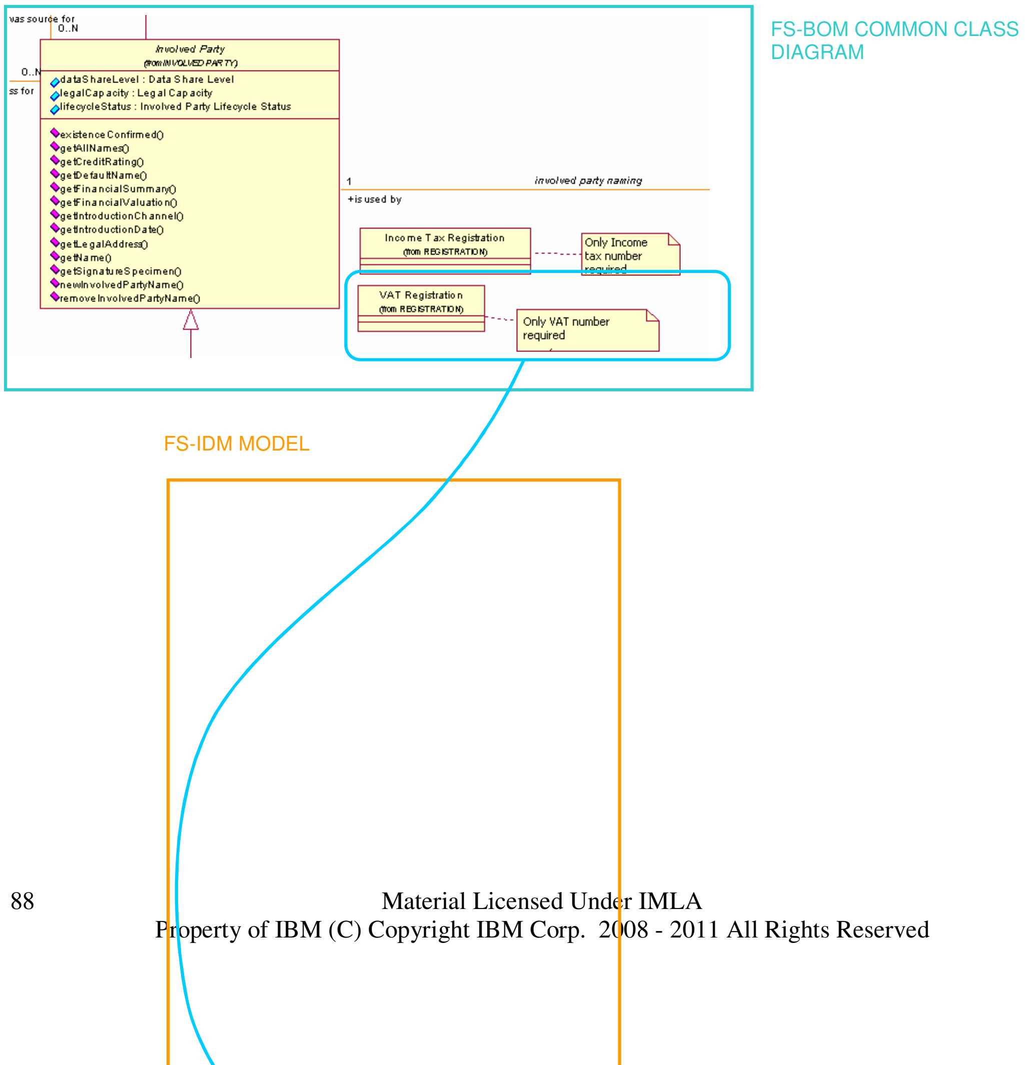
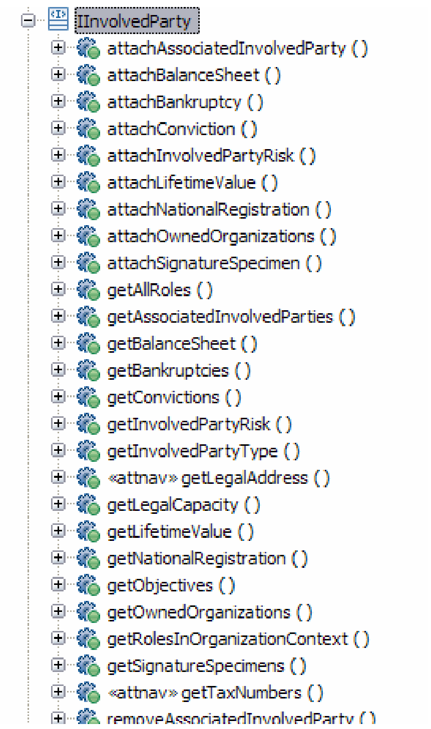


Figure 59. Interpreting whole types in common diagrams

Where a common class diagram shows a whole class related to the primary class, as in the diagram above, this can be modeled at design time using an aggregated relationship. This relationship will then specify that the source class wholly encompasses the target class. Aggregation can be by value or by reference, indicating unique ownership or shared ownership respectively.

BOT diagrams are not solely constructed from direct relationships like this however. A target that is at the end of a complex navigation, or a target only a part of which is considered common may be added to the diagram with an annotation describing the details of the requirement.





IInvolvedParty
attachAssociatedInvolvedParty ( )
attachBalanceSheet ( )
attachBankruptcy ( )
attachConviction ( )
attachInvolvedPartyRisk ( )
attachLifetimeValue ( )
attachNationalRegistration ( )
attachOwnedOrganizations ( )
attachSignatureSpecimen ( )
getAllRoles ( )
getAssociatedInvolvedParties ( )
getBalanceSheet ( )
getBankruptcies ( )
getConvictions ( )
getInvolvedPartyRisk ( )
getInvolvedPartyType ( )
«attnav» getLegalAddress ( )
getLegalCapacity ( )
getLifetimeValue ( )
getNationalRegistration ( )
getObjectives ( )
getOwnedOrganizations ( )
getRolesInOrganizationContext ( )
getSignatureSpecimens ( )
«attnav» getTaxNumbers ( )
removeAssociatedInvolvedParty ( )

Figure 60. Interpreting partial types in common diagrams

This analysis results in the addition of “attnav” operations to the IDM model. These are operations that “skip” over the complex navigations between types, and retrieve either all or just a part of the target classes’ details, effectively including those details as part of the data specification for the source type.

Where <<attnav>> operations overload the behavior of aggregating relationships, those relationships should be stereotyped as <<overloaded>>. Marking the relationship as “overloaded” prevents both paths being considered valid when describing the data pattern of the class, during generation. Consider the example below.

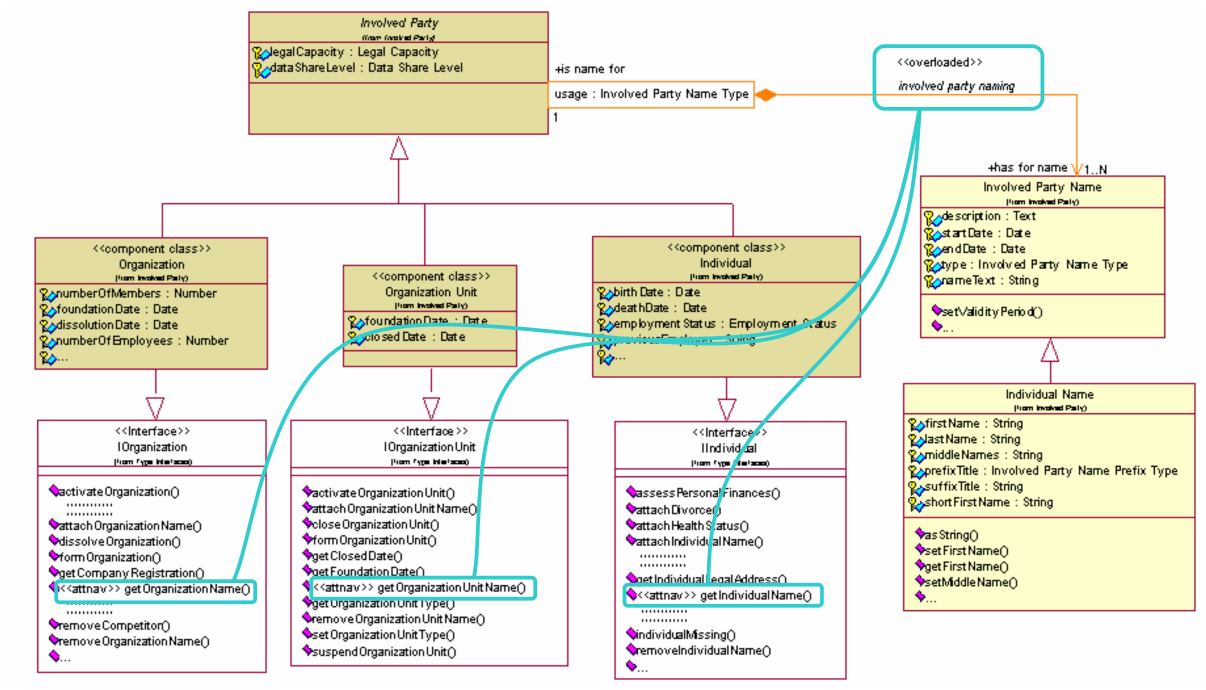


Figure 61. Association overloaded by "attnav" operations

Without the <<overloaded>> stereotype, the aggregation by value on the “involved party naming” association would result in the inclusion of Involved Party Name as part of the data pattern of Involved Party and its descendants. As the <<attnav>> operations also act to include Involved Party Name (or Individual Name in the case of <<attnav>> getIndividualName()), this would result in a duplication of data. To avoid this, where an association in the model is overloaded by <<typenav>> or <<attnav>> operations, that association is marked <<overloaded>>.

Operations stereotyped as “<<attnav>>” can also be used to model the requirement where a whole directly related class is considered common, instead of using an aggregated association. This is a design decision that centers on the desired flexibility over aspects of the data pattern, such as naming of the target element.

In this way all of the analysis captured within BOM, describing the desired common aggregates used across multiple service candidates, can be used to influence the design decisions of IDM resulting in defined type boundaries expressed in the model.

Migrating Type Customizations

The analysis within BOM will also include customizations to the class models. These changes must be propagated to IDM influencing the design of the data types associated with the service architecture.

New types added to BOM should be reviewed before being added to IDM.

The presence of a class in BOM indicates the presence of a requirement to describe a business concept. It is not necessarily the case that a separate design level construct is required to handle this requirement. Expansive sub-typing hierarchies in BOM are frequently “rolled-up” in IDM, in recognition of the fact that a single class can often be coded to handle multiple requirements. At design time, the decision as to when to roll up and when to retain sub-typing depends upon how different the business concepts in question really are.

Where two subtypes exhibit significantly different data requirements or operational requirements, they should be separated into two classes. Where subtypes exhibit only minor specializations, it may be appropriate to combine them into a single class at runtime. See the examples of collapsing class hierarchies in Appendix A.

Having decided that a type should be added to IDM, additional considerations may apply. For example, whether the type is a dependent or independent type and its associations to other types will influence the choice of component to which it is assigned. (Association classes are discussed in the later section on “Analyzing UML Associations”.) A dependent type will usually be assigned to the same component as the independent type that manages it.

If a dependent type is validly associated with more than one independent type, analysis of its UML associations with reference to component dependencies will help to determine where the new class should be located. For example, the dependent class Objective is the target of associations with the independent types Product and Involved Party. Component dependencies indicate that Objective be assigned to Involved Party. If Objective were to be assigned to the Product component, it would not be accessible to the Involved Party component and the association “role player objectives” would violate component dependencies.

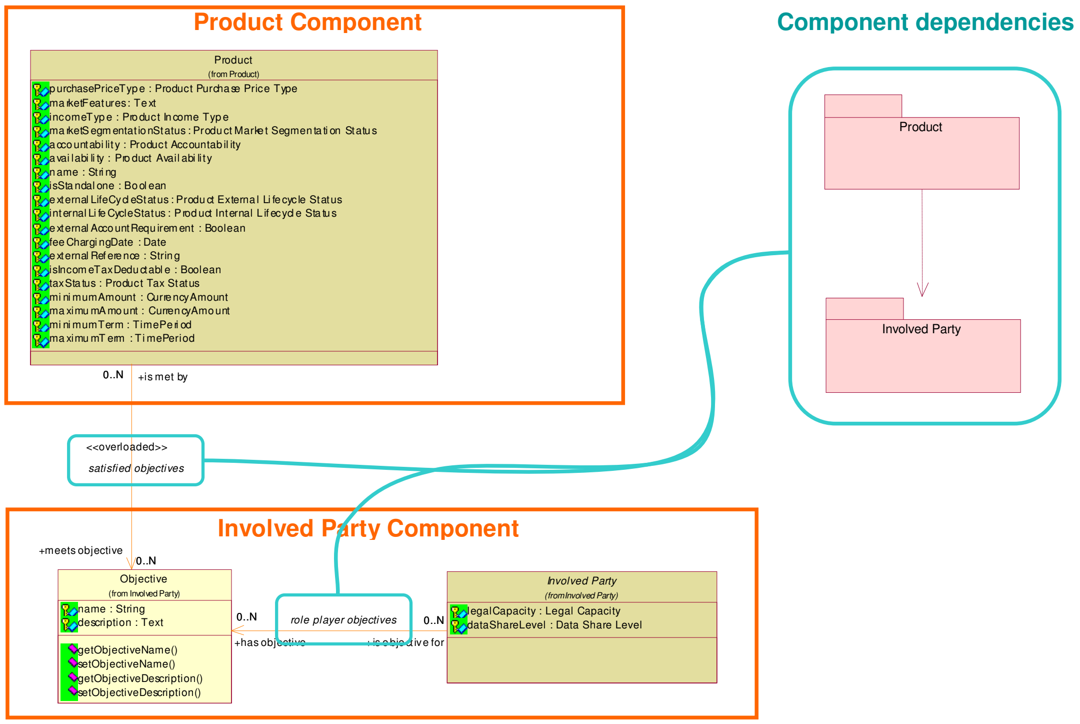

Product Component
Product
(from Product)
purchasePriceType : Product Purchase Price Type
marketFeatures : Text
incomeType : Product Income Type
marketSegmentationStatus : Product Market Segmentation Status
accountability : Product Accountability
availability : Product Availability
name : String
isStandalone : Boolean
externalLifeCycleStatus : Product External Lifecycle Status
internalLifeCycleStatus : Product Internal Lifecycle Status
externalAccountRequirement : Boolean
feeChargingDate : Date
externalReference : String
isIncomeTaxDeductable : Boolean
taxStatus : Product Tax Status
minimumAmount : CurrencyAmount
maximumAmount : CurrencyAmount
minimumTerm : TimePeriod
maximumTerm : TimePeriod
0..N
+is met by
<<overloaded>>
satisfied objectives
Component dependencies
Product
Involved Party
Involved Party Component
+meets objective
0..N
Objective
(from Involved Party)
name : String
description : Text
getObjectiveName()
setObjectiveName()
getObjectiveDescription()
setObjectiveDescription()
0..N
role player objectives
0..N
+has objective
+is objective for
Involved Party
(from Involved Party)
legalCapacity : Legal Capacity
dataShareLevel : Data Share Level

Figure 62. Dependent type component assignment

Assign independent types to components in such a way as to reduce inter-component dependencies. The best component will frequently be the component managing the super-type or peer types of the independent type. For example, a new type of Involved Party would be assigned to the Involved Party component. In this way, a type can be best described as being assigned to the component that defines its context.

However, certain independent types (e.g. sub-types of Classification) work closely with other independent types and are best assigned to the components containing these latter types. For example a new Classification sub-type would tend to be assigned to the component that manages the types to be classified. The class “Team” is a classification of Individuals and is assigned to the Involved Party component. The class “Product Group” is a classification of Products and is assigned to the Product component.

Take note of the inheritance pattern for the new class and check whether component dependencies are affected by its component assignment. A super-type of the new type may reside in a different component.

For example, some sub-types of Involved Party Role are assigned to the Arrangement component, while Involved Party Role itself is assigned to the Involved Party component. This implies that the Arrangement component must have access to the Involved Party component through its dependency hierarchy.

Similarly, the attribute types of the new type may reference types defined in other components. Generally these will be either primitive data types or enumerations. As before, the component containing these must be accessible, via the component dependency hierarchy, to the component in which the new type is defined.

Adding new independent types to IDM may imply addition of new operations to a manager interface of a component. For example, operations may be required to manage the new type or to bind it to other types.

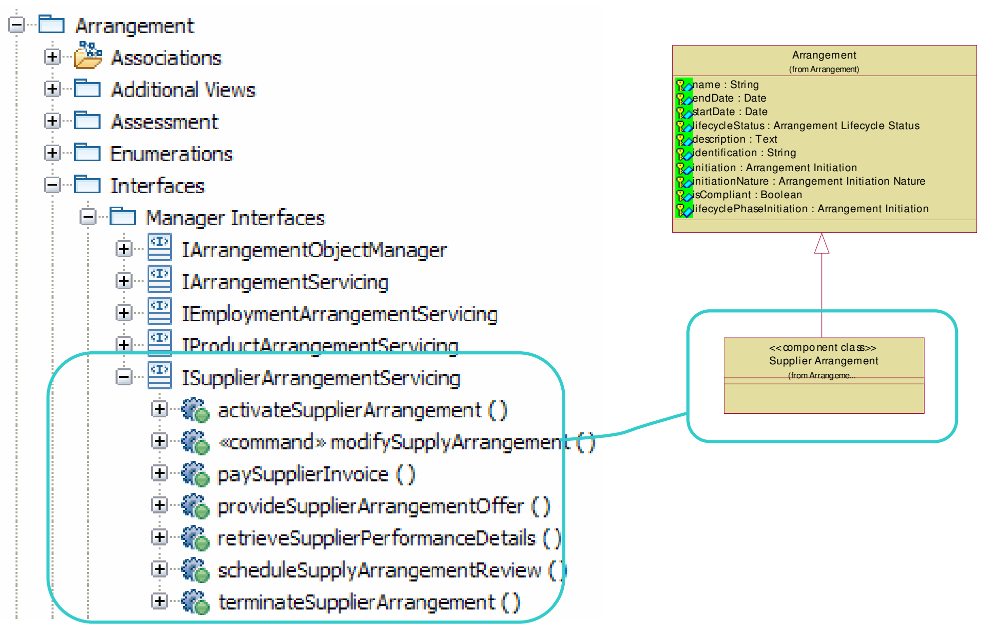

Arrangement
Associations
Additional Views
Assessment
Enumerations
Interfaces
Manager Interfaces
IArrangementObjectManager
IArrangementServicing
IEmploymentArrangementServicing
IProductArrangementServicing
ISupplierArrangementServicing
activateSupplierArrangement ( )
«command» modifySupplyArrangement ( )
paySupplierInvoice ( )
provideSupplierArrangementOffer ( )
retrieveSupplierPerformanceDetails ( )
scheduleSupplyArrangementReview ( )
terminateSupplierArrangement ( )
Arrangement
(from Arrangement)
name : String
endDate : Date
startDate : Date
lifecycleStatus : Arrangement Lifecycle Status
description : Text
identification : String
initiation : Arrangement Initiation
initiationNature : Arrangement Initiation Nature
isCompliant : Boolean
lifecyclePhaseInitiation : Arrangement Initiation
<<component class>>
Supplier Arrangement
(from Arrangeme...

Figure 63. Manager interface operations

New types may also imply new accessor operations on a remotely associated type. For example, a new classification can potentially be used by any independent type in the model. This is because a general purpose association between Business Model Object and Classification Value provides the ability for any Business Model Object sub-type to be classified by any Classification Value. However not all such bindings have business value. For example, it makes sense to assign Individuals, rather than Products, to a Team. This business idea is captured by providing specific operations on the appropriate interface of the component.

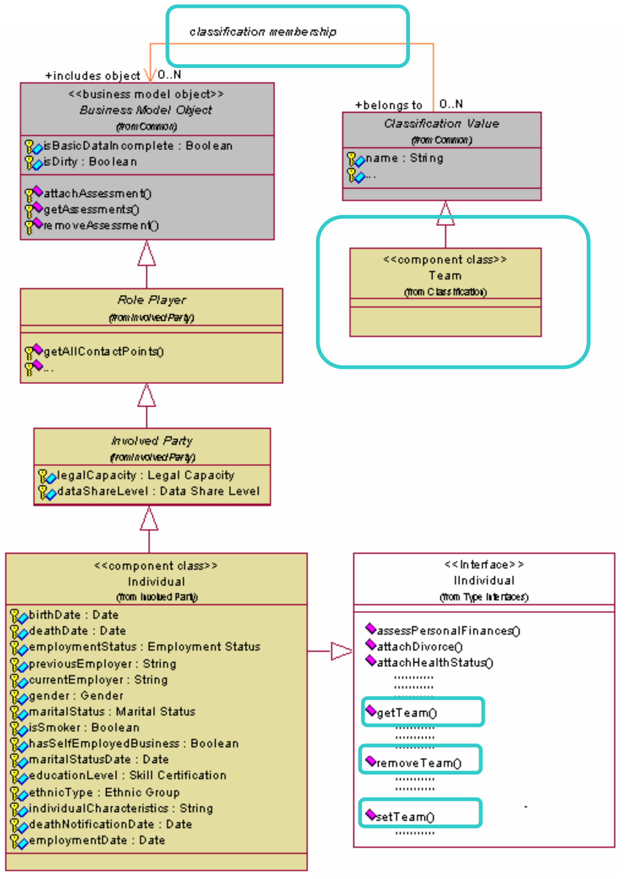


Figure 64. Operations implied by addition of new type

Attributes added to existing types in BOM will need to be added to the equivalent type in IDM. BOM to IDM mapping will help considerably here, allowing the rapid identification of the IDM construct that provides the design level representation of a BOM construct. However, assignment of responsibilities (operations) within an interface design model remains a design task and the logical break-down of interfaces within the model must be given due consideration.

When customizations of attributes are made to IDM, sub-typing structures should also be reviewed. Sub-typing decisions are made on the basis of the inherent differences between subclasses. Adding new attribution will alter this balance, and may result in the introduction of further sub-typing that was previously excluded from IDM.

Attributes added to IDM will usually be protected to the class on which they reside. This implies that access to these attributes must also be added through accessor operations on dependent types or type interfaces. Accessor operations should not be added automatically to the model. Design considerations should apply to whether attributes are set as a group, or individually, whether attributes can be retrieved but never set, or vice versa. Naming standards in IDM distinguish between setting values on attributes of a type and creating a link to another type (this is generally an "attach" operation e.g. attachIndividualName()) – see Appendix B for examples.

Operations added to BOM capture requirements such as navigational or management responsibilities. These operations will affect the definition of type interfaces within IDM, resulting in the exposure of new fine-grained services in the model.

If these operations embody a behavioral area that is semantically distinct from other operations, then it may be better to add a new interface, rather than extending an existing one.

Occasionally an interface grows to an unmanageable size, at which point it may be necessary to split it. It is important that this is done in a meaningful way. Examine the existing interface to determine useful behavior and use this to guide selection of a new interface.

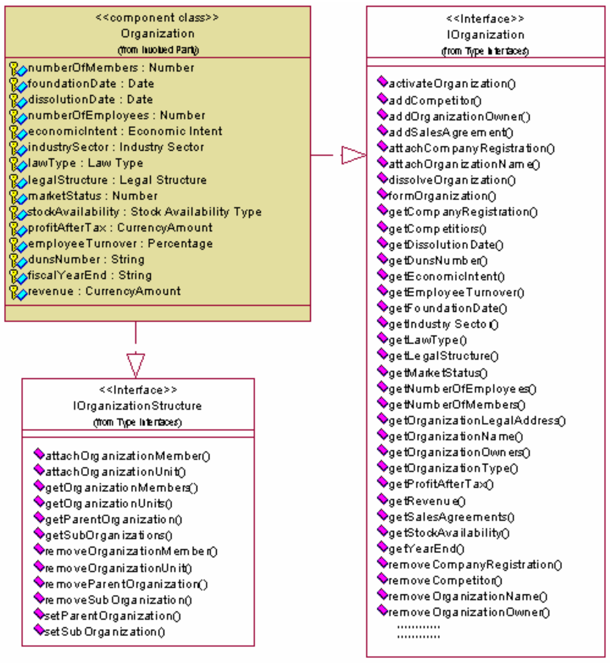


Figure 65. Functionality split across interfaces

Analyzing UML Associations

UML associations added to BOM will affect the structure of IDM. Associations added in BOM simply capture the requirement that one type relates to another in a defined way. At design time we must consider the directionality of these associations, whether the relationship is a simple association or an aggregation, and any qualification of that association.

Associations may have additional data provided via association classes. These must be added to IDM, attached to the relevant association and assigned to a component.

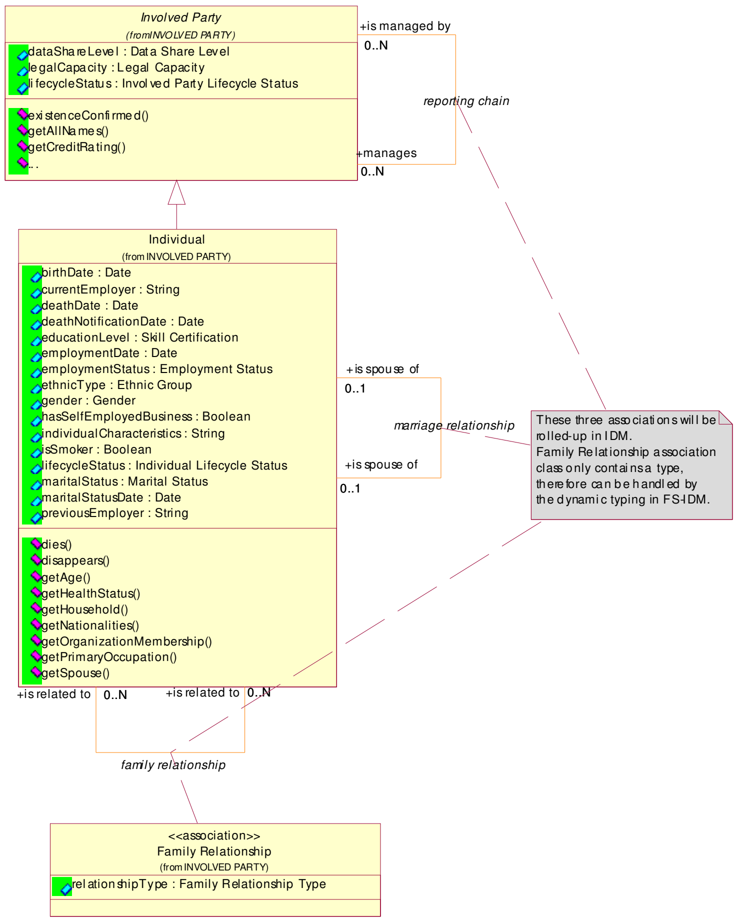
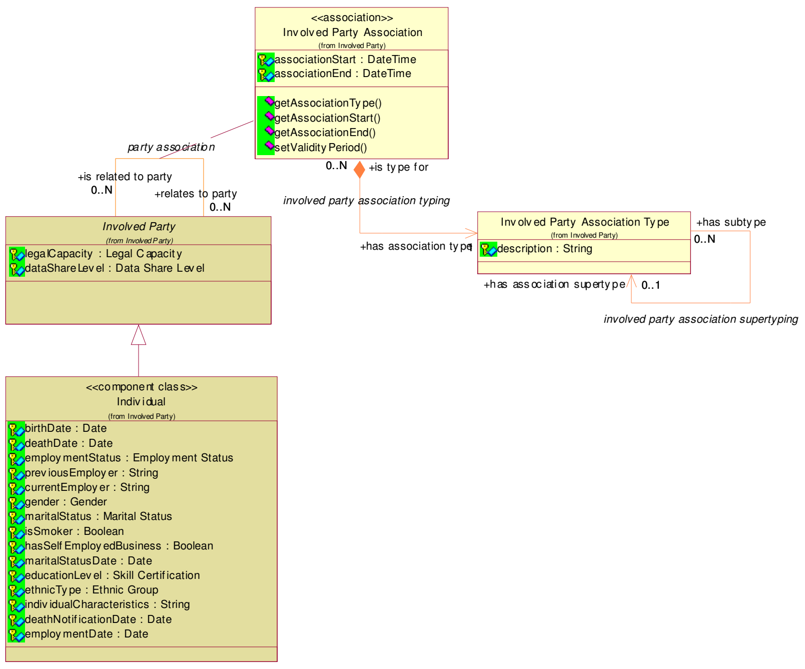

Involved Party
(from INVOLVED PARTY)
dataShareLevel : Data Share Level
legalCapacity : Legal Capacity
lifecycleStatus : Involved Party Lifecycle Status
existenceConfirmed()
getAllNames()
getCreditRating()
...
+is managed by
0..N
reporting chain
+manages
0..N
Individual
(from INVOLVED PARTY)
birthDate : Date
currentEmployer : String
deathDate : Date
deathNotificationDate : Date
educationLevel : Skill Certification
employmentDate : Date
employmentStatus : Employment Status
ethnicType : Ethnic Group
gender : Gender
hasSelfEmployedBusiness : Boolean
individualCharacteristics : String
isSmoker : Boolean
lifecycleStatus : Individual Lifecycle Status
maritalStatus : Marital Status
maritalStatusDate : Date
previousEmployer : String
dies()
disappears()
getAge()
getHealthStatus()
getHousehold()
getNationalities()
getOrganizationMembership()
getPrimaryOccupation()
getSpouse()
+is spouse of
0..1
marriage relationship
+is spouse of
0..1
These three associations will be rolled-up in IDM. Family Relationship association class only contains a type, therefore can be handled by the dynamic typing in FS-IDM.
+is related to
0..N
+is related to
0..N
family relationship
<<association>>
Family Relationship
(from INVOLVED PARTY)
relationshipType : Family Relationship Type

Figure 66. BOM associations

Multiple associations between the same source and target class may be collapsed to a dynamically typed general purpose association. Associations with association classes may not be captured by a general purpose association, unless the content of such an association class can be captured meaningfully at the dynamic typing level.

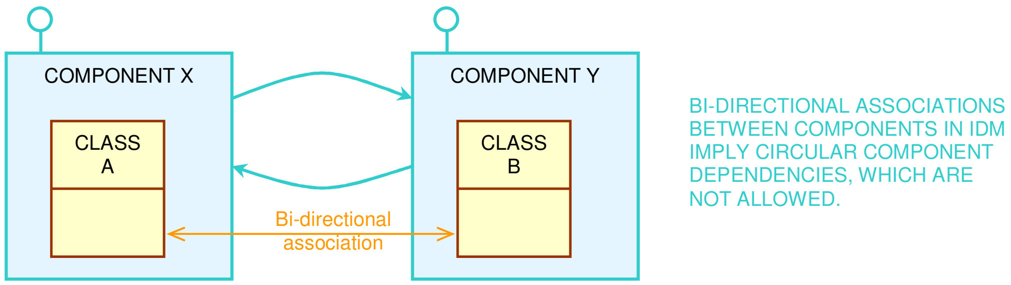


<<association>>
Involved Party Association
(from Involved Party)
associationStart : DateTime
associationEnd : DateTime
getAssociationType()
getAssociationStart()
getAssociationEnd()
setValidityPeriod()
party association
+is related to party
0..N
+relates to party
0..N
0..N
+is type for
involved party association typing
Involved Party Association Type
(from Involved Party)
description : String
+has subtype
0..N
+has association type
+has association supertype
0..1
involved party association supertyping
Involved Party
(from Involved Party)
legalCapacity : Legal Capacity
dataShareLevel : Data Share Level
<<component class>>
Individual
(from Involved Party)
birthDate : Date
deathDate : Date
employmentStatus : Employment Status
previousEmployer : String
currentEmployer : String
gender : Gender
maritalStatus : Marital Status
isSmoker : Boolean
hasSelfEmployedBusiness : Boolean
maritalStatusDate : Date
educationLevel : Skill Certification
ethnicType : Ethnic Group
individualCharacteristics : String
deathNotificationDate : Date
employmentDate : Date

Figure 67. IDM associations with dynamic typing

Component dependencies are key to the structure of IDM, defining how the services required by the enterprise can be segregated across a set of logical units that interact in defined patterns. For this reason, special care must be taken when defining relationships between types that reside in different components of IDM. Typically classes are protected within the scope of their host component, exposing their capabilities through interfaces. This means that the target of an association may be an interface and not a class at all.

Bi-directional relationships between concepts that now reside in different components imply a bi-directional (or circular) dependency between the components involved, which results in non-linear component architecture, and violates the meta-model of IDM. Analysis that relies on bi-directional associations must be reviewed at design time, to ensure that the linear nature of IDM component dependencies is preserved.



COMPONENT X
CLASS A
COMPONENT Y
CLASS B
Bi-directional association
BI-DIRECTIONAL ASSOCIATIONS BETWEEN COMPONENTS IN IDM IMPLY CIRCULAR COMPONENT DEPENDENCIES, WHICH ARE NOT ALLOWED.

Figure 68. Bi-directional associations in IDM

For example, in “Figure 68” above, a bi-directional association between class A and class B implies that component X has access to component Y, and that component Y also has access to component X. This will result in an illegal circular dependency between components X and Y in IDM.

This can be resolved as illustrated in the diagram below, by separating the implementation of type A from the behavior of type A, through encapsulation. Defining a type interface iA, which is available much lower down the component stack, and not providing an implementation of this interface as class A until much higher in the stack results in linear component dependencies. Note also that the single BOM association has resulted in two IDM associations.

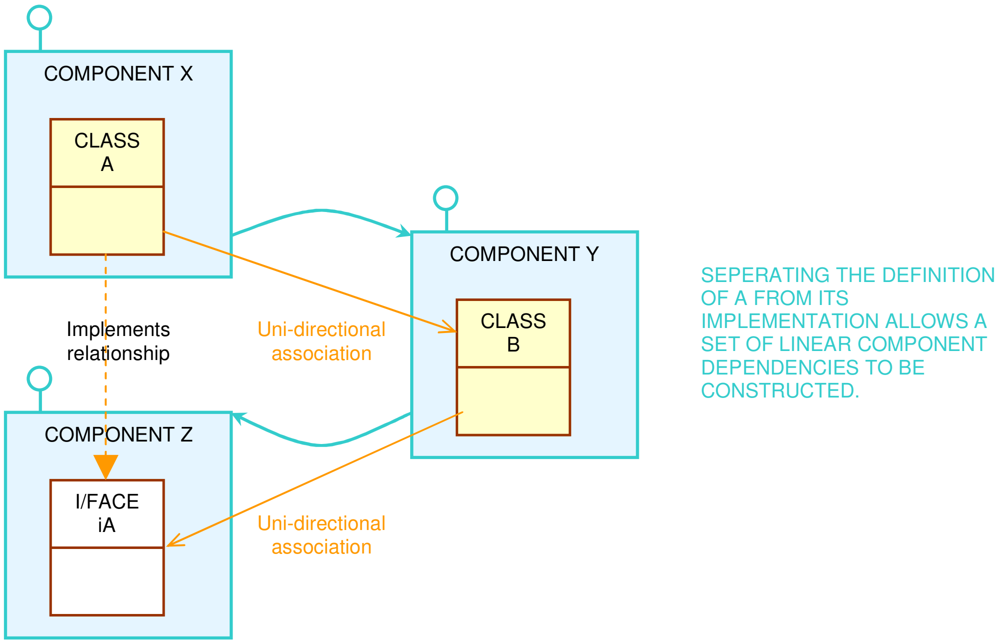

COMPONENT X
CLASS A
COMPONENT Y
CLASS B
SEPERATING THE DEFINITION OF A FROM ITS IMPLEMENTATION ALLOWS A SET OF LINEAR COMPONENT DEPENDENCIES TO BE CONSTRUCTED.
Implements relationship
Uni-directional association
COMPONENT Z
I/FACE iA
Uni-directional association

Figure 69. Using encapsulation for linear dependencies

As discussed under the topic of boundary of type, associations should also be reviewed in the light of BOT diagrams, to determine where aggregation relationships are required to support reusable types across business services.

Associations may also be qualified at design time to specify that only one instance of the association can exist for a particular data pattern. For example, analysis that specified that an “Involved Party” could only have one “Involved Party Name” for a given value of “Involved Party Name Type” (Trading Name, Legal Name…) would result in the relationship between “Involved Party” and “Involved Party Name” being qualified by “Involved Party Name Type”.

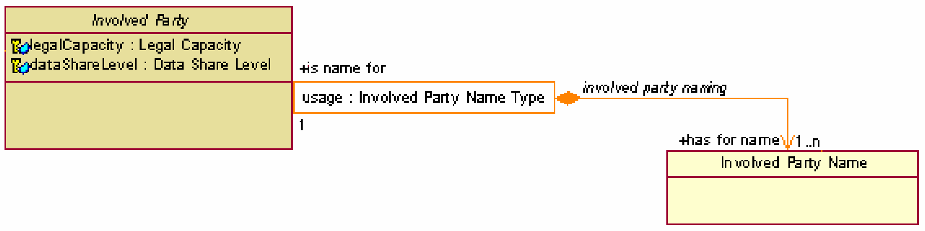

Involved Party
legalCapacity : Legal Capacity
dataShareLevel : Data Share Level
+is name for
usage : Involved Party Name Type
1
involved party naming
+has for name
1..n
Involved Party Name

Figure 70. Qualified relationships in IDM

Assigning Classes to Components

Newly added types in IDM must be assigned to a component, in line with the meta-model rules of IDM. A class cannot be assigned to more than one component, and where a class definition is used within multiple components this implies a dependency relationship between the classes involved.

Chapter 5. Realization and Implementation

The key acceleration provided by IBM's Industry Models is within the early (Identification and Specification) phases of the SOMA method. While the models still play a role within the Realization and Implementation phases, this role is significantly less, and as such these two phases on SOMA are presented in a single chapter.

5.1. Realization

The realization phase of SOMA takes the output of service specification, and applies a set of activities designed to enable decisions about the realization of those services in software. For example, within the realization phase we answer key questions such as the data integration or service composition patterns that are best suited to implementing the services we have identified. As such, most of this phase of SOMA is not directly impacted by the use of the Industry Models, however some key decisions are highlighted below that affect the flow of Industry Models content into the Implementation Phase.

5.1.1. Information Realization

Information realization discusses the mapping of information services to the technical patterns and data services that support them. This is important in the context of the Industry Models as we wish to identify the technical environments to which information service definitions should be published during the Implementation phase.

Select Information Strategic Pattern
Key to the realization of information services is the identification of the data integration patterns that best suit the services that were produced during the Specification phase. Example data integration patterns are data federation, extract, transform and load (ETL) and data centralization. For detail on the relevance and application of these patterns, see the Information Discipline papers listed under 'Related Documentation' at the start of this paper.

Define data source plan
A related realization decision is whether implementation of information services requires the construction or modification of existing data sources, or indeed the use of a packaged application. Mapping of proposed information services to existing or proposed data sources can guide the specification of these services, and identify gaps or other issues with the data sources in use before implementation.

5.1.2. Refine Component Specification

A review of the final component designs specifications resulting from the specification phase can identify further opportunities to optimize these components based on the planned implementation landscape. In particular, existing systems or packaged applications should be considered and the component specifications refined accordingly. These refinements of the component specifications and their relationship to the target environment can inform decisions as to the realization of services as WSDL, SCA Modules, BPEL modules or other structures.

5.1.3. Establish Realization Decisions

At this stage of the Realization phase, final decisions are made as to how the specified services and the software that supports them will be realized. While IBM's Industry Models are largely agnostic of these realization decisions, there are commonly recurring patterns. Service contracts are typically realized through expression as WSDL interfaces with XSD types describing the parameters of these services. WSDL and XSD can be created from the model in a number of ways, one of which is through the use of the IMA generators. These generators read the IDM model, and infer XSD type structures and WSDL request responses from it. A number of generation options are available within this utility, which is documented separately.

Service implementations will be realized through a wide range of methods depending on the nature of the service. For example, simple services will often be wrappers around the capability of existing systems, perhaps using the data and service mapping capabilities of WebSphere Integration Developer (WID), deploying as SCA modules. Information services may be realized through a range of patterns as highlighted in the Information Discipline papers listed at the start of the document. Complex service choreography may merit BPEL modules, while complex decision services may be supported through the use of rules engines or analytics.

These realization decisions are made at this stage in the SOMA method in order to guide downstream service implementation.

5.2. Implementation

As IBM's Industry Models deliberately retain technology independent structures, this section of the SOMA method contains only references to tooling capabilities that are of interest when using the models.

Following key steps are performed to leverage Industry Model during SOMA Implementation:

- Export WSDL/XSD definitions of these IDM services using the generator plug-in
  - WSDL includes Request/Response Types
  - XSD built based on aggregations and stereotypes
- Implement components for example using Java Design Model for custom development

Specifically, implementation guidance is available on the following areas:

5.2.1. Develop WSDL/XSD for service interfaces

As indicated above, WSDL/XSD generation can be performed for service contracts within the IDM model through the IMA Generators. These generators identify operations within the models stereotyped as <<command>> and generate WSDL request and response pairs for each. The types that support these services are expressed through a set of XSD files, corresponding to the structural components of the IDM model. Each XSD file contains the types defined within that component. Note that the transformation between the IDM model and the types in these XSD files is not direct. In fact the source for these XSD types is often the type interfaces within the IDM model. These type interfaces act as denormalised specifications of the highly normalized types in the model. Through these type interfaces, the designer can specify the precise data set, sourced from the underlying class model that is to be presented to the service consumer. This allows multiple denormalisations of the same class model depending on the specific needs of service consumers, without compromising the underlying class model of IDM.

5.2.2. Implement Components

The component model of IDM is a platform independent model (PIM). This model can be transformed into platform specific models is required for the design on technology specific components. One example of this transformation is the use of the IDM2JDM transformation, which transforms the IDM model into a Java Design Model (JDM). This resultant model can then be modified as required, and used to generate Java code, for example through the use of Rational Application Developer (RAD).

Chapter 6. Conclusions

The Industry Models provide a structured set of models that support the scoping, analysis and design challenges facing financial institutions. The SOMA based roadmap detailed within this document will guide all parties through the use of these models to achieve defined results. The following diagram shows synergy between SOMA and the industry models during the development life cycle.

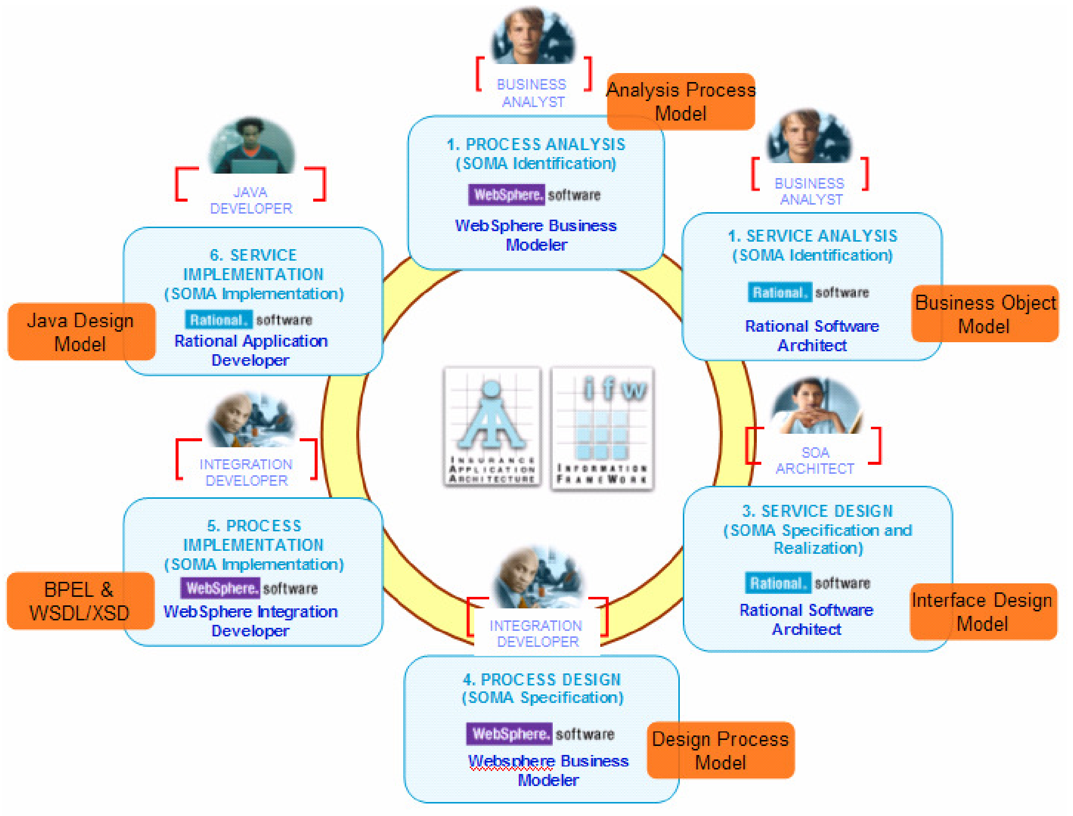

BUSINESS ANALYST
Analysis Process Model
1. PROCESS ANALYSIS (SOMA Identification)
WebSphere software
WebSphere Business Modeler
BUSINESS ANALYST
1. SERVICE ANALYSIS (SOMA Identification)
Rational software
Rational Software Architect
Business Object Model
JAVA DEVELOPER
6. SERVICE IMPLEMENTATION (SOMA Implementation)
Rational software
Rational Application Developer
Java Design Model
INTEGRATION DEVELOPER
SOA ARCHITECT
3. SERVICE DESIGN (SOMA Specification and Realization)
Rational software
Rational Software Architect
Interface Design Model
5. PROCESS IMPLEMENTATION (SOMA Implementation)
WebSphere software
WebSphere Integration Developer
BPEL & WSDL/XSD
INTEGRATION DEVELOPER
4. PROCESS DESIGN (SOMA Specification)
WebSphere software
Websphere Business Modeler
Design Process Model
Insurance Application Architecture
Information FrameWork

Figure 71. Applying SOMA to IBMs Industry Models

The Industry Models do not however make the challenges in analysis and design disappear. Depending on the internal structure of an organization, the very concept of reuse of analysis and design can prove extremely difficult. Organizations can be limited by organizational, political, cultural and other constraints, which can mean real changes must take place in order to facilitate individual project efforts to share information effectively.

The potential benefits of reuse usually outweigh the difficulty involved however, and the initial investment in a new way of looking at scoping, analysis, design and development can yield real contributions towards the organization IT consolidation efforts.