Genus Apps ®
A Technology Overview

Introduction

Genus Apps® (often shortened *Genus*) is a high-productivity or rapid software application development technology. The technology is model-driven without any need for programming or scripting. This significantly reduces the effort required to design, build, deploy and manage your business apps[1] as compared to traditional programming or configurable products.

In the software engineering world, modeling has a rich tradition, dating back to the earliest days of programming. Traditional innovations have focused on notations and tools that allow users to express system perspectives of value to software architects and developers. These notations and tools are then manually mapped into program code and compiled for a particular operating system. The data about business objects and logic is interwoven with the program code, only to be changed through reprogramming.

In the Genus approach, the meta-data (the model) used to describe the business objects and logic have sufficient detail to enable Genus to run from the meta-data itself without the need of programming or code generation. To us, a model is not something incomplete or inaccurate, but distilled knowledge and a way of structuring a domain. It is easier to change the model than program code, making it a lot faster to adapt your apps to your changing business processes. It also enhances the quality of your software apps by reusing and not rebuilding common functionality. Also, unlike traditional programming, you do not need to worry about changes in the underlying platforms, like Microsoft Windows, iOS and Android, since Genus take cares of the adaptation to such changes.

Gold
Microsoft Partner
Microsoft

Most customers use Genus to add functionality to their old apps, or build bridges between them, utilizing the data and concepts in the old apps. Some customers build entirely new apps using our technology, often replacing one or more old systems. More and more customers use Genus to prototyping, which holds the advantage that there is no need to throw away the prototype if one decides to develop it to a final solution – one just continues to develop the prototype.

Fundamentals

From an academic perspective, Associate Professor Hallvard Trætteberg at the Norwegian University of Science and Technology (NTNU), has described Genus using these words:

Genus Apps is a model-based app development tool and runtime for rich data-intensive domains, typical of modern businesses. The models cover data using an Entity Relationship-like language, business logic with both declarative and procedural elements and user interface with forms and rich visual components (graphs, Gantt charts and diagrams), including structure, layout and binding to underlying business data. Apps are deployed on Microsoft Windows, on web (all major browsers) and as apps (all major mobile devices and tablets).

By our own words, we would like to add that Genus is a general-purpose modeling language: it is a visual language for any domain. It is a tool for specifying models, where these models are the primary and only artifact, and the models are interpreted at runtime. Models are specified both graphically and

[1] *Please note that we are using the terms "app" and "application" interchangeably. As you will experience throughout this white paper, Genus Apps is not constrained to tablet or mobile apps, but covers all platforms, including desktop and web apps.*

textually and Genus covers both static and dynamic models. Genus Apps is *codeless* and belongs to the *no code/low code*[2] type of high productivity development tools.

Genus is *not* about model driven architecture (MDA) and *not* about code generation.

Perspectives

In the following, the technology aspects of Genus are described from four perspectives:

- Functional: The technology as seen from a business analyst's point of view.
- Architectural: The technology as seen from a technical architect's point of view.
- Integrational: The technology as seen from a system integrator's point of view.
- Implementational: The technology as seen from a project leader's point of view.

Contents

[2] *There are currently some discrepancies concerning the terms "no code" and "low code" in the marketplace and among IT research firms. Currently Genus Apps belongs to both the no code and low code "camps" depending on which parts of Genus Apps you use and your flavor of definition.*

1 Functional perspective

The functional perspective is concerned with the technology as seen from a business analyst's point of view.

1.1 No programming involved

Genus does not require any programming or scripting. In fact, it is *not* possible for you to insert program code into Genus, thus making Genus a true model driven app development tool.

From a functional point of view this is a major point, since this raises the level of abstraction, narrowing the gap between your business analysts' specifications and the app.

The introduction of model driven app development bears similarities with the introduction of spreadsheets in the late 1980's. The spreadsheets raised the level of abstraction, in their particular field of application, by getting away from writing such programs entirely. Nobody today would use a traditional programming tool (such as C++, C#, Java etc.) to write an app to perform spreadsheet-style processing. So, the spreadsheet applications have moved your focus away from specifying *how* (programming) to calculate something - to specifying *what* to calculate.

The underlying historical trend is clear - we are moving away from procedural (how) and towards declarative (what). HOW means saying how, step by step, the work is to be *done*; WHAT just means saying what the work to be done *is*.

The advantage is obvious, you become significantly more productive as the app development work gets done much more easily and much more quickly. Our customers are experiencing reduction in app development time by an order of magnitude or more.

1.2 User interface

Genus Apps for Web, our web client, is based on standard browser technology. The web user interface supports most browser user interface features. Web Apps are not truly native, but have the advantage of the same look and feel across all major mobile and tablet devices.

Genus Apps for Desktop, our Microsoft Windows client, has adopted the Microsoft usability concepts aligned with the Microsoft Office interface.

1.3 Concepts

Modeling an app using Genus requires an understanding of the concepts involved. The most important concepts are briefly described below.

1.3.1 App model

An app model is a formalized description of a software app's user interface, logic and data, expressed using Genus Apps. The behavior and appearance of your Genus app is determined by the content of your app model, i.e. the model is the app.

Note that our use of the term "model" does not imply that a model is some form of simplification. To us a model is distilled knowledge and a way of structuring a domain.

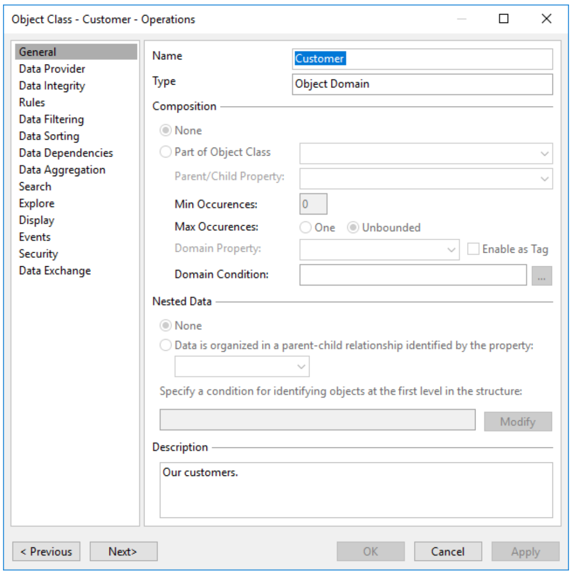
1.3.2 Data set

A data set is a collection of data for a single unit of business, being for example a country.

1.3.3 Class and composition

Object Classes express static descriptions of data and behavior. Data for an Object Class is defined by adding properties. An Object Class describes your specific objects, such as Customer or Person.

Object composition is a way and practice to combine simple objects into more complex ones. For example, the objects *wheel*, *steering wheel*, *seat*, *gearbox* and *engine* may have no functionality by themselves. But if you compose them, they may form an *automobile* object. That is, an object which has a higher function (or greater than the sum of its parts).


Object Class - Customer - Operations

Name: Customer

Type: Object Domain

Composition

● None

○ Part of Object Class

Parent/Child Property:

Min Occurences: 0

Max Occurences: ○ One ● Unbounded

Domain Property: □ Enable as Tag

Domain Condition:

Nested Data

● None

○ Data is organized in a parent-child relationship identified by the property:

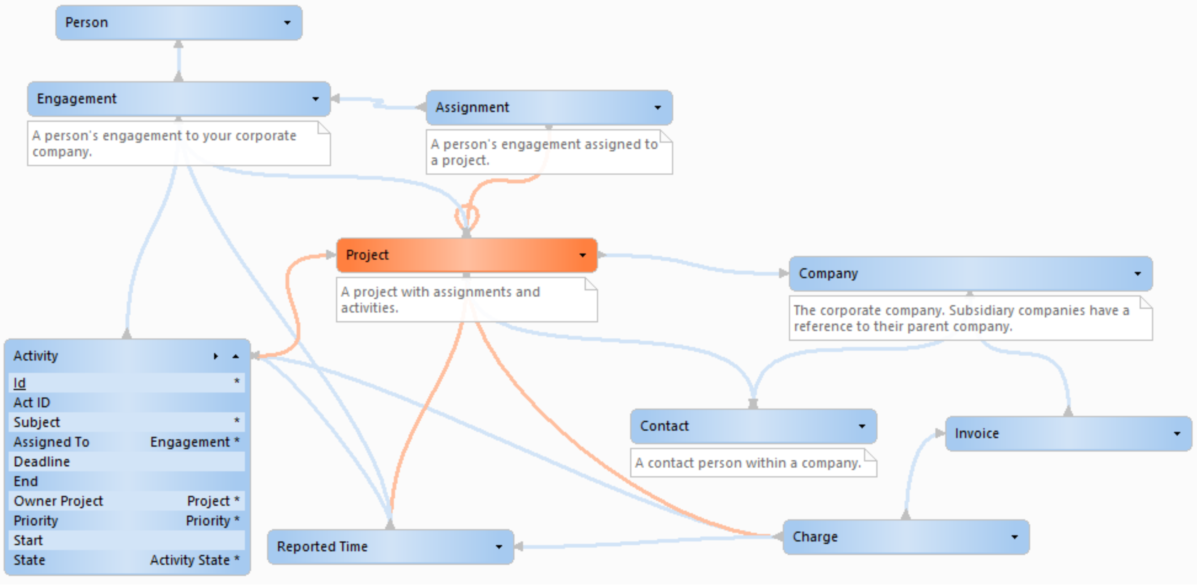
Specify a condition for identifying objects at the first level in the structure:

Description

Our customers.

A specialized (child) Object Class may be based on a general (parent) Object Class using a Complex association (see chapter 1.3.4 for more information).

Object Classes may be connected to other object classes through property references and visualized (and edited) in Genus Data Diagrams.



A person's engagement to your corporate company.

A person's engagement assigned to a project.

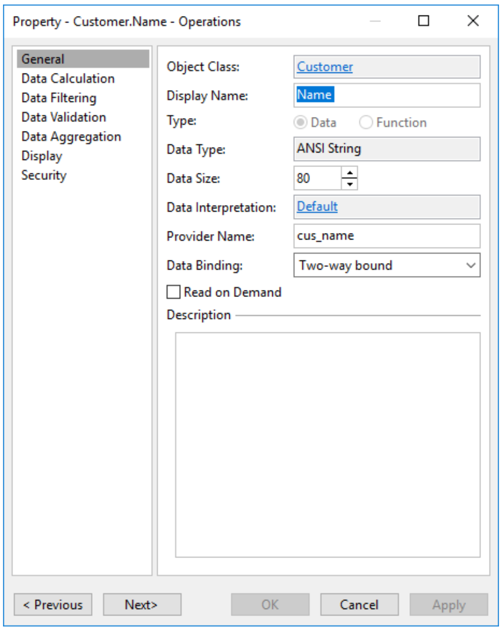
A project with assignments and activities.

The corporate company. Subsidiary companies have a reference to their parent company.

A contact person within a company.

Activity	
Id	*
Act ID	
Subject	*
Assigned To	Engagement *
Deadline	
End	
Owner Project	Project *
Priority	Priority *
Start	
State	Activity State *

1.3.4 Object class property



Object Class:	Customer
Display Name:	Name
Type:	● Data ○ Function
Data Type:	ANSI String
Data Size:	80
Data Interpretation:	Default
Provider Name:	cus_name
Data Binding:	Two-way bound

□ Read on Demand

Description

Properties express static descriptions of data for an Object Class. A property can be defined as a Data or Function property. Values for properties of type Data are stored in the database. Values for properties of type Function are calculated using a formula or a RDBMS expression.

The kind of data a property can hold is defined by its data type and data interpretation.

Data types are categorized in simple data types, complex data types and complex associations. Simple data types include Boolean, Date, Integer, Internet URL, Float, Binary Large Object (BLOB) and Unicode String, and more.

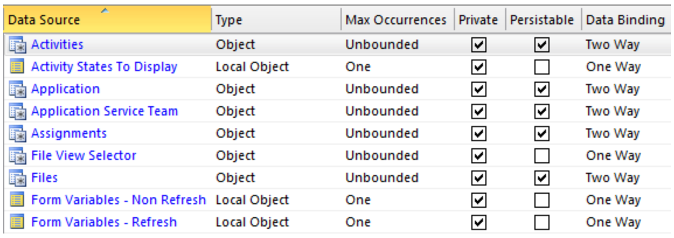
Complex data types are your objects, for example Customer or Person, as defined by your Object Classes.

Complex association is used when the property can represent *any* Object Class within a predefined set of Object Classes. A common use of complex association is an Object Class containing a single outbound reference to a varying number of Object Classes. For example, an Object Class which represents a document can be attached to an employee, a customer, a project or an activity. Complex associations resemble extensions (inheritance or is-a relationships) in Unified Modeling Language (UML) class diagrams.

1.3.5 Data sources

Data sources are your sources of information and made up of fields and groups. Data sources are on a higher level than Object Classes, since Data sources may use Object Classes.

The kind of data stored in a field or group is defined by the data type.



Data Source	Type	Max Occurrences	Private	Persistable	Data Binding
Activities	Object	Unbounded	☑	☑	Two Way
Activity States To Display	Local Object	One	☑	□	One Way
Application	Object	Unbounded	☑	☑	Two Way
Application Service Team	Object	Unbounded	☑	☑	Two Way
Assignments	Object	Unbounded	☑	☑	Two Way
File View Selector	Object	Unbounded	☑	□	One Way
Files	Object	Unbounded	☑	☑	Two Way
Form Variables - Non Refresh	Local Object	One	☑	□	One Way
Form Variables - Refresh	Local Object	One	☑	□	One Way

A field can store data for both simple and complex data types. A group is, by definition, a complex data type. For example, if you add a data source of type Customer, the data type for the top-level group in the data source is Customer. Within a data source, a group can be defined as repeating, meaning that data for the group occurs more than once. For example, all Products in a given Product Category.

There are several types of data sources, like Object (for example Customer), XML Document, Report and File.

Using the File data source, you may directly create, read, modify and delete objects like File Folders, General Files, Mail Messages, Outlook Contacts (vCard) or Outlook Calendar Items (iCal).

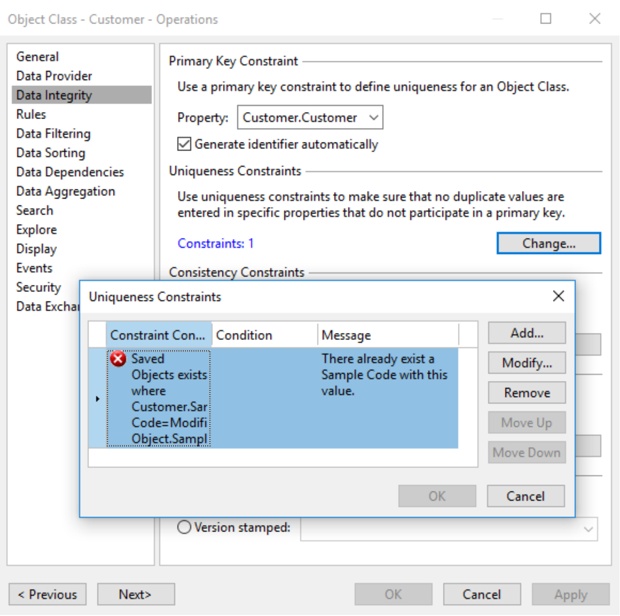
1.3.6 Data integrity

You can use Primary Key Constraints to define uniqueness for an Object. You can also define rules to ensure the accuracy of data and its conformity to its expected value by using uniqueness constraints, consistency constraints and delete constraints.

A primary key or a unique key comprises a single property or a set of properties. A Primary Key Constraint states that two objects cannot have the same value (or combination of values) for those properties.

Use of more general Uniqueness Constraints ensures that no duplicate objects are entered in specific properties that do not participate in a primary key.

Use of Consistency Constraints ensures that values entered for specific properties satisfy certain requirements for correctness and accuracy.



Use a primary key constraint to define uniqueness for an Object Class.

Property: Customer.Customer

☑ Generate identifier automatically

Use uniqueness constraints to make sure that no duplicate values are entered in specific properties that do not participate in a primary key.

Constraints: 1

Uniqueness Constraints

Constraint Con...	Condition	Message
Saved Objects exists where Customer.Sar Code=Modifi Object.Sampl		There already exist a Sample Code with this value.

○ Version stamped:

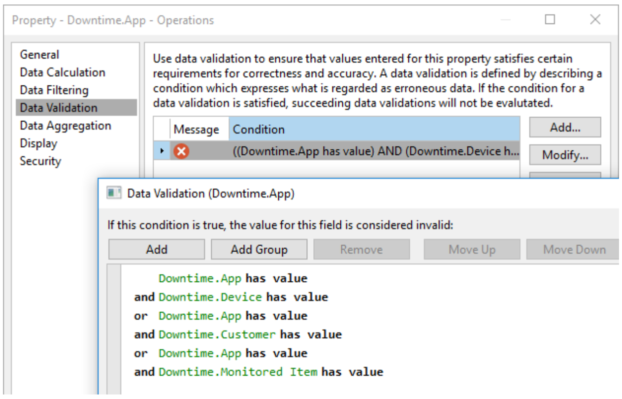
You can define a special set of rules or constraints used for restricting access to delete an object. A Delete Constraint is defined by specifying a condition, which expresses a state for an object which cannot be deleted. You can add multiple delete constraints to an Object Class.

1.3.7 Data validation

Data validation can be used to ensure that values entered for a Property satisfies certain requirements for correctness and accuracy.

Several types of data validation are supported:

- **Required value.** The value for a Property cannot be blank.
- **Data type validation.** The value for a Property has to be of a particular type, such as a whole number or a date.
- **Condition-based validation.** Defined by describing a condition, which expresses what is regarded as erroneous data. For example, that the start date for a project not can occur after the end date.



Use data validation to ensure that values entered for this property satisfies certain requirements for correctness and accuracy. A data validation is defined by describing a condition which expresses what is regarded as erroneous data. If the condition for a data validation is satisfied, succeeding data validations will not be evalutated.

Message	Condition
	((Downtime.App has value) AND (Downtime.Device h...

If this condition is true, the value for this field is considered invalid:

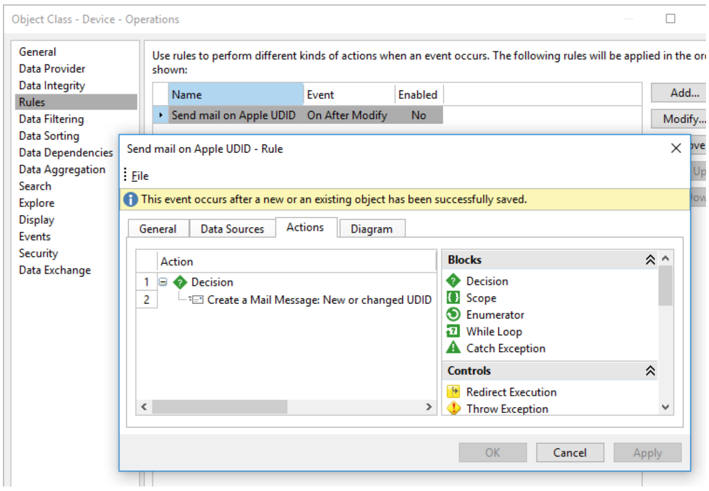
```
    Downtime.App has value
and Downtime.Device has value
or  Downtime.App has value
and Downtime.Customer has value
or  Downtime.App has value
and Downtime.Monitored Item has value
```

1.3.8 Rule

Behavior is defined by adding rules that contains the knowledge on how to respond to different events. A rule can be triggered by events like On After Modify, On Delete, On Before Create etc. A rule will fire a sequence of actions as defined in chapter 1.3.9.

A rule can be attached to an object class, for instance making it possible to notify people by email (Send a mail) when a new activity is created (On After Create) on your specific project.

Rules can be bound to a specific Data Set.



Use rules to perform different kinds of actions when an event occurs. The following rules will be applied in the or shown:

Name	Event	Enabled
Send mail on Apple UDID	On After Modify	No

This event occurs after a new or an existing object has been successfully saved.

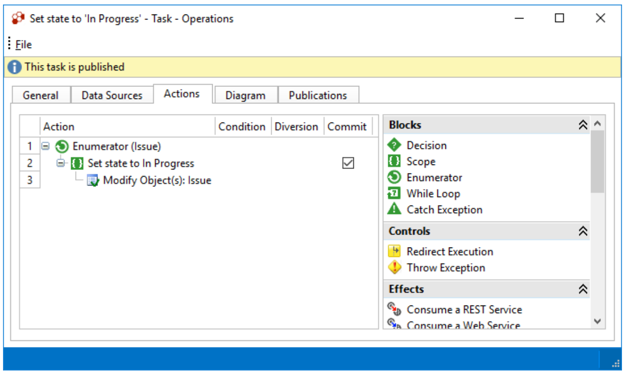
	Action
1	Decision
2	Create a Mail Message: New or changed UDID

1.3.9 Task

A task is a sequence of *actions* which again consist of effects. The definition of an Action follows in the section below.

Tasks can be published to the user interface (data sources, forms, tables), but also to standard verbs such as *Open*, *New*, *Run*, *Print* and *Export*. Tasks can be attached to an object class through a rule.

Tasks can be bound to a specific Data set.

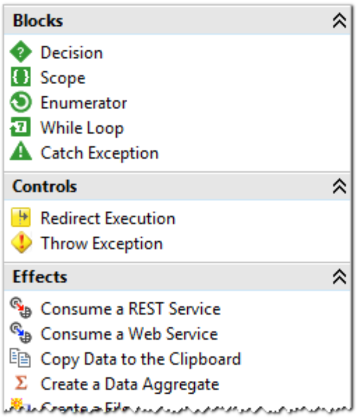


This task is published

	Action	Condition	Diversion	Commit
1	Enumerator (Issue)			
2	Set state to In Progress			☑
3	Modify Object(s): Issue			

1.3.10 Action and effect

An action is used to group a set of effects. An effect is used to perform a specific task within a sequence of actions. The following effects exist: Consume a REST Service, Consume a Web Service, Copy Data to the Clipboard, Create a Data Aggregate, Create a File, Create a Mail Message, Create Object(s), Delete Object(s), Export Data, Import Data, Invoke a File, Merge Data to a Document, Merge Documents, Modify a User Account, Modify Object(s), Open a Form, Print Object(s), Read Object(s), Run a Data Extract, Run a Program, Run a Report, Run a Task, Set a Field's value, Show a Confirmation Message, Show a Message, Sort Object(s), Transform XML Data, Upload a File.

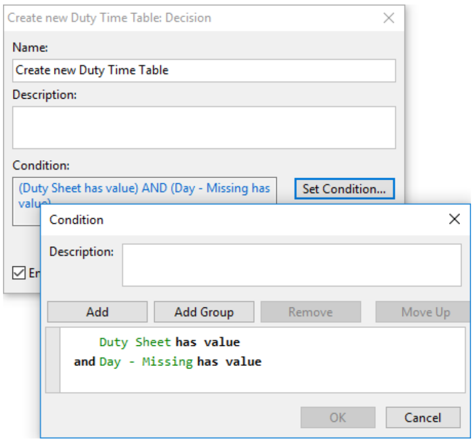


Blocks are used to organize and control sets of actions. A block can contain effects, controls, and other blocks. Each block type has distinct characteristics to allow control of flow, transactions, iterations, and exception handling. The following blocks exist: Decision, Scope, Enumerator and Catch Exception.

Controls are used to change the normal execution flow, by transferring control to other actions. The following controls exist: Redirect Execution and Throw Exception.

Actions are used by Tasks, Agents, Web Services and Rules to define a sequence of tasks that should be performed during execution. To perform a specific task, for example importing data from a file, an effect supporting the requested task is added to the action.

An action is executed based on a condition.



Create new Duty Time Table: Decision

Name: Create new Duty Time Table

Description:

Condition: (Duty Sheet has value) AND (Day - Missing has valu...

Condition

Description:

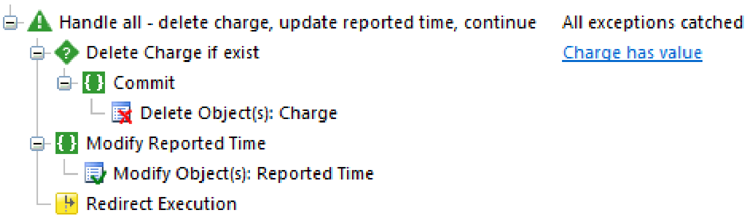
```
  Duty Sheet has value
and Day - Missing has value
```

1.3.11 Conditions

A condition consists of comparisons, which evaluate to TRUE or FALSE. A comparison states a logical relationship between two values, for example *Person.Name = "Peter Keating"*, or *100 > 200*. A condition is often a logical statement about the state or existence of an object, or the relationship between objects.

1.3.12 Exception handlers

During execution of an action, there may occur errors that could result in a disruption of the normal execution flow. These errors are often referred to as exceptions, and can be handled by adding exception handlers to the action.
Exception handlers are special steps which are dedicated to resolve error situations, and are only executed if an error occurs.



Handle all - delete charge, update reported time, continue   All exceptions catched
  Delete Charge if exist   Charge has value
    Commit
      Delete Object(s): Charge
  Modify Reported Time
    Modify Object(s): Reported Time
  Redirect Execution

There are several types of exceptions, and an exception handler can be defined to handle one or more of these types of exceptions. In an action, there can be more than one exception handler for each type of exception, and when an exception occurs, the execution of the action will jump to the first relevant exception handler following the error. If the error occurs in a task, which is called from a web service, an agent, a rule, or another task, the execution of the action will jump to the first relevant exception handler even if this is located outside the task itself.

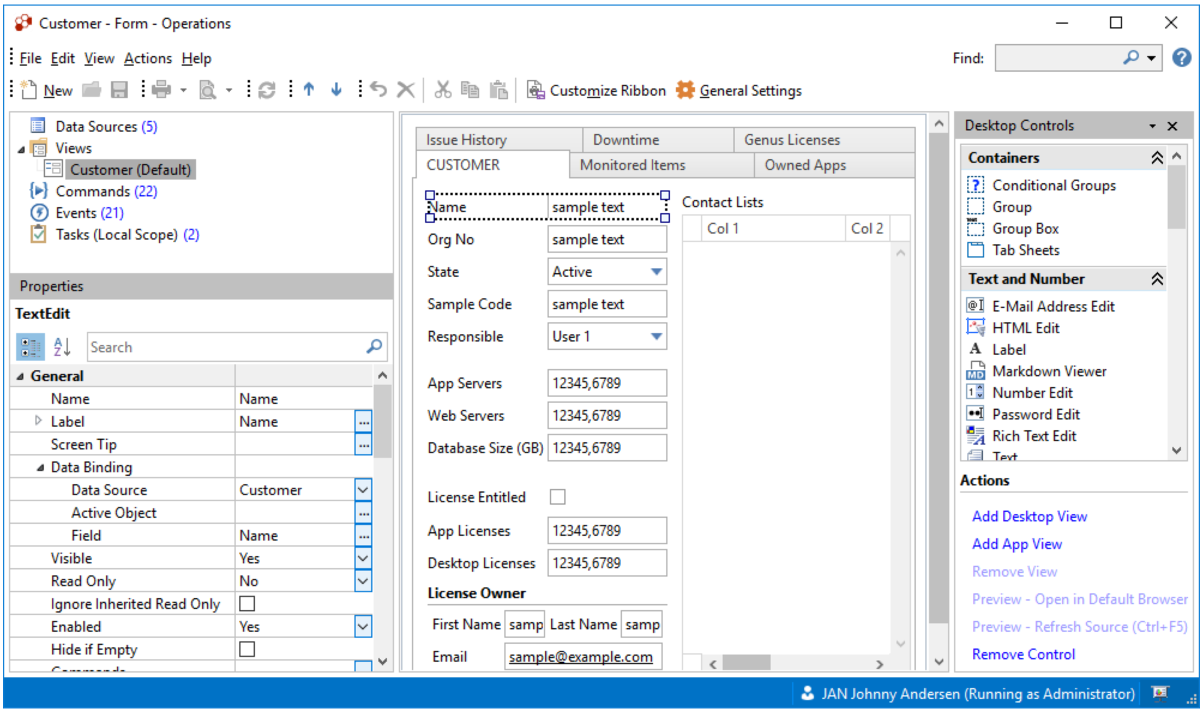
The different effects can throw several types of exceptions. Examples of exceptions are On Transaction Error, On Concurrency Error, On Data Constraint Violation, On Uniqueness Constraint Violation, On File Read Error, On Send Mail Error and On Web Service Communication Error. See the figure below for more information.

Effect/Exception	Transaction			Logic							File						OS			Mail		Web Service				REST		Security		Other		
	OnTransaction Cancelled by User	OnTransaction Error	OnTransaction Time Out	OnConsistency Check Violation	OnCardinality Violation	OnConcurrency Error	OnData Constraint Violation	OnDelete Constraint Violation	OnExpression Evaluation Error	OnUniqueness Constraint Violation	OnFile Exists	OnFile Not Found	OnFile Read Error	OnFile Record Rejected	OnFile Write Error	OnPath not Found	OnApplication Not Installed	OnInvalid Program Exit Code	OnProgram Execution Error	OnMail Merge Error	OnSend Mail Error	OnWeb Service Communication Error	OnWeb Service Endpoint not Found	OnWeb Service Fault	OnWeb Service Message Security Error	Onhttp Communication Error	Onhttp Content Type Mismatch	OnInsufficient Security Permissions	OnModify User Account Error	OnObjects Read Error	OnPrint Error	OnCustom Error
Consume a REST Service					•		•		•				•													•	•					
Consume a Web Service									•													•	•	•	•							•
Copy Data to the Clipboard									•						•																	
Copy Permissions									•																			•				
Create a Data Aggregate	•	•	•		•		•		•	•																		•				
Create a File									•				•		•		•															
Create a Mail Message									•												•											
Create Objects	•	•	•		•	•	•		•	•																		•				
Delete Objects	•	•	•	•		•		•	•																			•				
Export Data to a File									•		•		•		•	•																
Import Data from a File					•		•		•			•		•		•																
Invoke a File									•			•	•		•		•															
Merge Data to an MS Office Document									•								•		•	•												
Modify a User Account									•																				•			
Modify Objects	•	•	•			•	•		•	•																		•				
Modify Permissions									•																			•				
Open a Form	•								•				•															•				
Print Objects									•																						•	
Read Objects	•				•				•																					•		
Run a Program									•									•	•													
Run a Task	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•	•
Set a Field's Value									•																							
Show a Confirmation Message									•																							
Show a Message									•																							
Throw a Custom Exception									•																							•
Transform XML Data									•																							

1.3.13 Form

A Form is a visual representation of one or more of your Object Classes, exposing the properties of your Object Classes, including any relationships with other Object Classes.

A Form allows the user to enter or modify data in your app, and constitutes as such the user interface of the objects in your app. A Form presents bound data in various highly customizable formats using a range of controls, such as text edits, number edits, labels, combo box edits, buttons, tables, charts, graphs, etc. The layout is easily customized by using container controls, such as groups, group box and tab sheets.



General	
Name	Name
Label	Name
Screen Tip	
Data Binding	
Data Source	Customer
Active Object	
Field	Name
Visible	Yes
Read Only	No
Ignore Inherited Read Only	☐
Enabled	Yes
Hide if Empty	☐

Name	sample text
Org No	sample text
State	Active
Sample Code	sample text
Responsible	User 1
App Servers	12345,6789
Web Servers	12345,6789
Database Size (GB)	12345,6789
License Entitled	☐
App Licenses	12345,6789
Desktop Licenses	12345,6789

License Owner

First Name samp Last Name samp

Email sample@example.com

Contact Lists

Col 1	Col 2

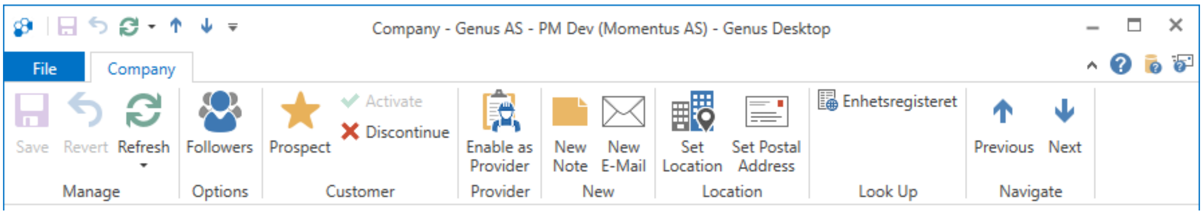
All the data that appears in a Form is stored in the form’s data sources. A control that present data is bound to a data source or to a field within a data source. You can present data from multiple data sources within the same form, and make them aware of each other. For example, if you click on a person in a table, edit boxes can appear with the data for that person to be viewed or modified. You can also add data sources which are not bound to any controls. For example, you can add a data source containing exchange rates, and then use these rates for calculating an amount in different currencies.

When you design a form, you must consider how to lay out and configure the controls in the form layout designer, to achieve a user-friendly interface. This can sometimes be a challenging task, as the form may contain data from many different data sources, which may or may not be dependent on each other. Generally, you should organize the controls in a logical manner, group related controls, and only show relevant controls. The primary objective is always that the users can easily see and access the data and functions they require at any time.

Forms are available on both web and desktop (Microsoft Windows). A specific Form is made available across platforms by targeting different views within the Form towards each platform.

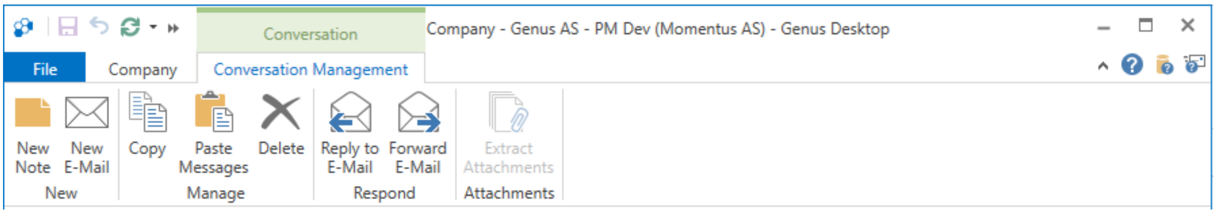
1.3.14 Ribbon

The ribbon is a command bar that organizes the features of an application into a series of tabs at the top of the application window.



The ribbon user interface increases discoverability of features and functions, enables quicker learning of the application, and makes users feel more in control of their experience with the application. The ribbon replaces the traditional menu bar and toolbars.

Context tabs (see example below) appear with relevant commands as the user focus the control the commands are assigned to.



The ribbon allows users to quickly browse and understand the available commands, and find the command needed for completing the task in very few clicks. The combination of symbol and text enhance recognition and discoverability of commands.

The ribbon is customized for each Form and is completely empty in newly created forms. This flexibility allows a clean and minimalistic design, where key commands are highlighted and unused commands can be removed.

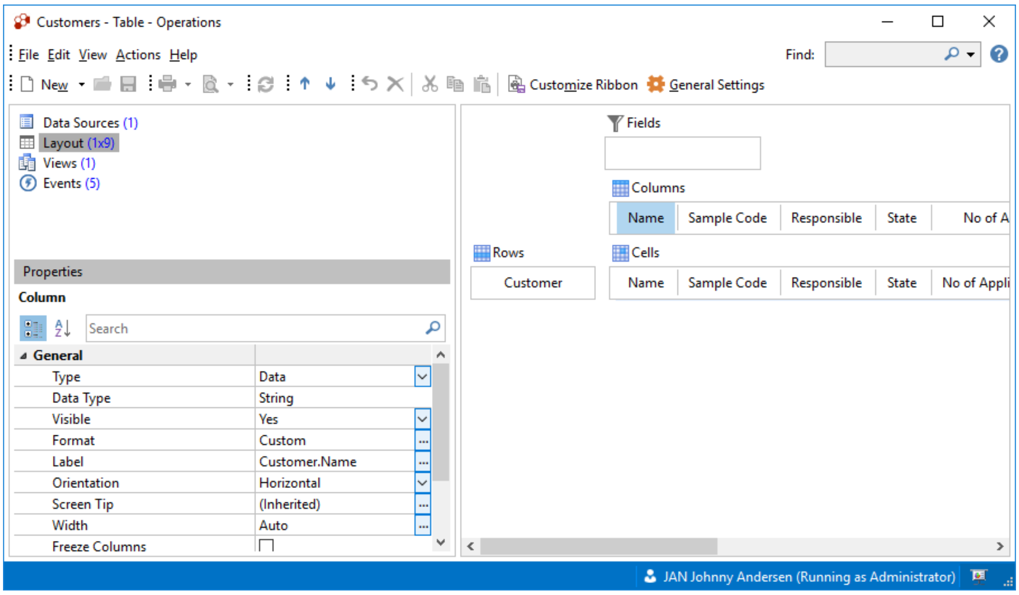
Features including tabs, context tabs, sections, large and small symbols with captions, toggle buttons and dropdowns creates an instructive and educational view of the commands in the form. The order and size of the commands indicates their importance, and grouping in sections can be used to describe related commands or sequences of commands. Enabling and visibility is used to visualize the state of commands. Screen tips and disabled tips assists the user.

The ribbon can be minimized to free up window space. The quick access toolbar, consisting of the most used commands, is always visible at the top of the window.

1.3.15 Table

A Table presents bound data in a tabular format, where columns represent data fields and rows represent objects. Numerous data management and layout customization features are supported.

All the data that appears in a Table are stored in the Table's data sources. Each row template in the Table are bound to a data source, and cell templates are bound to fields within a data source. You can present data from multiple data sources within the same Table. That is, when you add a new column to a Table, simply bind a field to the cell template for the column for each row template. You can also add data sources which not are bound to any row- or cell templates. For example, you can add a data source containing currency exchange rates, and then use these rates for calculating an amount in different currencies.



General	
Type	Data
Data Type	String
Visible	Yes
Format	Custom
Label	Customer.Name
Orientation	Horizontal
Screen Tip	(Inherited)
Width	Auto
Freeze Columns	□

When you design a Table, you must consider how to lay out and configure the elements in the Table. Generally, you should organize the elements so that users who fill it out can enter data in a logical manner. The Table layout designer helps you achieve this goal. In the Table layout designer, you can add and group columns, add row templates, and bind row- and cell templates to your data sources. Numerous data management and presentation features can be customized for each of the elements in the Table.

By designing different views of your Table, you can offer users diverse ways to look at data. For example, you might create a special view that is optimized for printing, or you might create a high-level summary view to eliminate some of the details in a complex table template.

Enabling of built-in commands such as New and Open, and actions and analysis available to users in the Task Pane, can be customized for each Table.

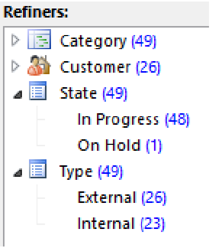
1.3.16 Search and refine

Genus supports searching through your structured business data, utilizing indexes in your database.

Genus also supports full text search of data and attached documents using your database's full text search functionality.

With Refiners, you may filter your data by point & click *related* data, e.g. reduce the number of found persons by clicking skills or states. You may even add such criteria in a sequence, change the sequence, remove criteria from the sequence and negate selected criteria.

Unlike traditional internet search, Genus adheres to security rules, and only searches among the information a user has access to.



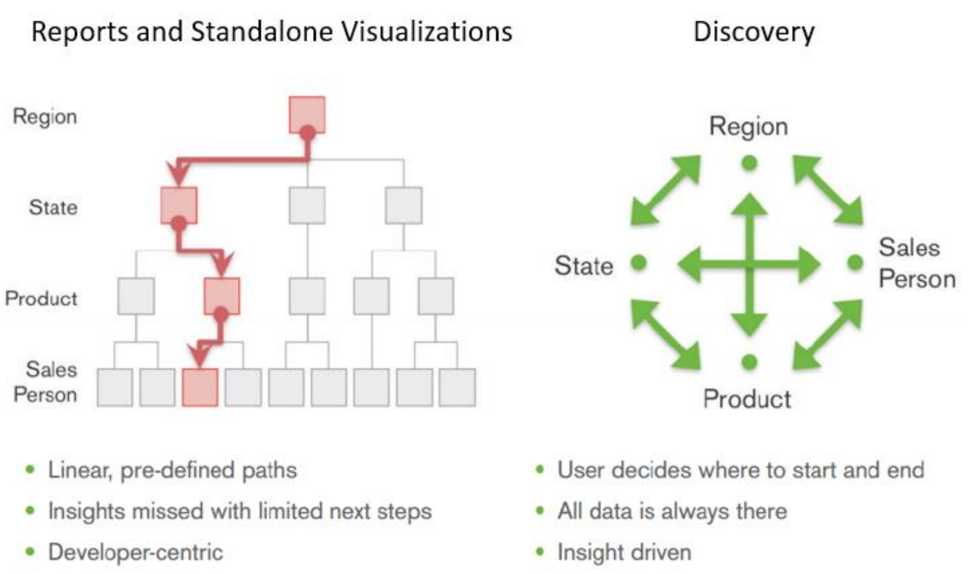
Refiners:

- Category (49)
- Customer (26)
- State (49)
  - In Progress (48)
  - On Hold (1)
- Type (49)
  - External (26)
  - Internal (23)

1.3.17 Analysis and data mart

Analysis presents data and connections from Data Marts, in an associative and intuitive manner. The Analysis presents data in various controls, like lists, fields, and charts. Select one or more data elements, and instantly discover which other data elements are related according to the selection, and which are not related.

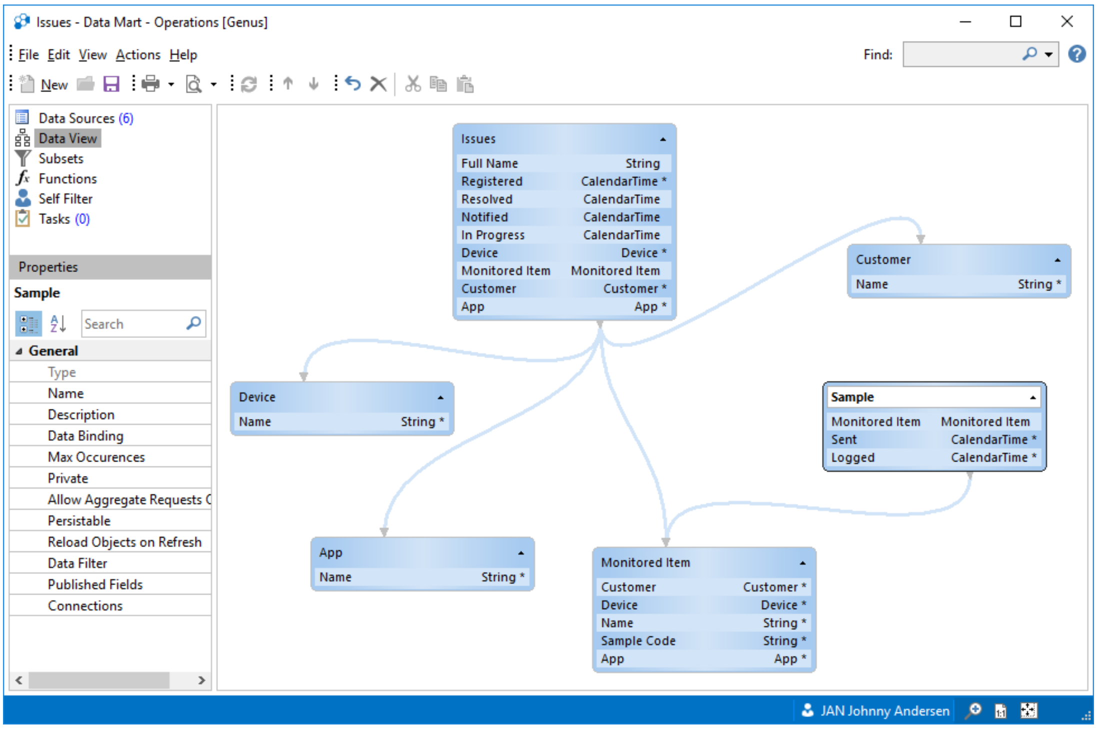
Analysis belongs to the Business Discovery class of products, which allow users to follow their “information scent” or “train of thought” when navigating through data. As seen in the figure below (adapted from [1]), there are no pre-specified drill-down patterns, and users decide where to start and end. Grouping, joining, and calculations are performed on-the-fly by using high-performance, in-memory technologies.



- Linear, pre-defined paths
- Insights missed with limited next steps
- Developer-centric

- User decides where to start and end
- All data is always there
- Insight driven

A Data Mart contains a set of data sources, fields, and connections, which initially is a subset of the classes, properties, and connections in your app model. New data sources, fields, and connections are then added to the Data Mart as necessary. The Data Mart also contains information on how the data sources are loaded with data, which allows for loading subset of data, and creating Data Mart specific aggregates. The Data Mart also provides ad-hoc aggregation of data, based on the needs of the Analysis.



Sample

General
- Type
- Name
- Description
- Data Binding
- Max Occurences
- Private
- Allow Aggregate Requests C
- Persistable
- Reload Objects on Refresh
- Data Filter
- Published Fields
- Connections

Issues	
Full Name	String
Registered	CalendarTime *
Resolved	CalendarTime
Notified	CalendarTime
In Progress	CalendarTime
Device	Device *
Monitored Item	Monitored Item
Customer	Customer *
App	App *

Customer	
Name	String *

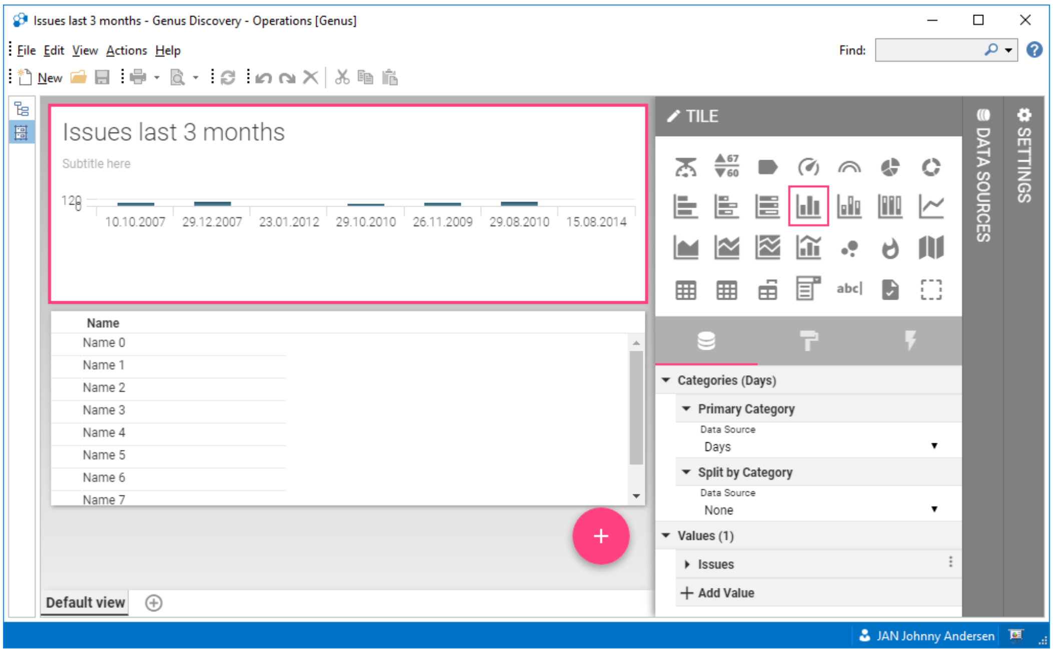
Device	
Name	String *

Sample	
Monitored Item	Monitored Item
Sent	CalendarTime *
Logged	CalendarTime *

App	
Name	String *

Monitored Item	
Customer	Customer *
Device	Device *
Name	String *
Sample Code	String *
App	App *

An Analysis is connected to a Data Mart, and the end user can visualize the data in the data sources by adding fields to the view. Various controls can visualize and arrange the data, like lists, fields, grids, charts, tab sheets, and groups. When a data element visualized in a control is selected, several things happens. The Data Mart refines all data sources based on the selected data elements. All affected ad-hoc aggregates are updated. The visualization of all data elements changes to reflect changes in aggregates, and whether they are connected to the selected data or not.



The Analysis is available on both web and desktop. The data in an Analysis may be merged with Microsoft Excel.

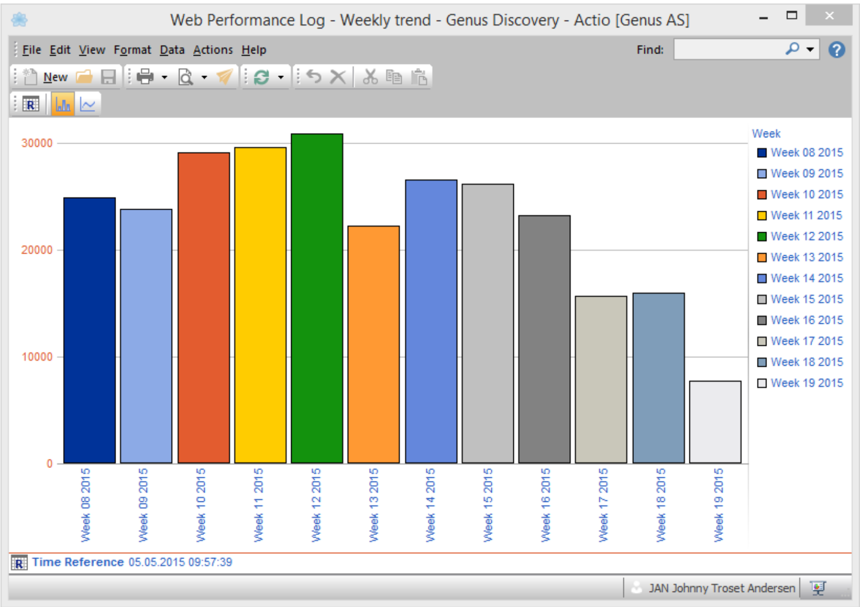
Compared to *Reports*, Analysis offers more flexibility when it comes to the *end user's* ability to change the visualization of data. Such flexibility is not always required or possible (e.g. due to performance / volumes), making Reports a natural alternative.

Analysis is an integral part of Genus Apps, i.e. the functionality and the technology is not based on a third-party vendor like Qlik, Tableau or similar.

1.3.18 Reports

A Genus Report may consist of numeric facts called measures which is categorized by classes (dimensions) in your app, allowing for complex analytical hoc queries with a rapid execution time. Reports are used for Business Intelligence (BI) purposes, such as sales revenue by products and/or departments, or by associated costs and incomes. Some customers use Genus Apps mainly for BI purposes.

Reports may use your data directly, or use aggregations of your data. Aggregations are built from your base data by changing the granularity on specific dimensions and aggregating up data along these dimensions. Genus support building of such aggregations through the Create a Data Aggregate effect. Note that Genus is *not* an Extract, transform, and load (ETL) platform.



Time Reference 05.05.2015 09:57:39

Your results may be presented in documents using the Merge Data to a Document effect.

1.3.19 Mail merge

Mail Merge is a feature in Microsoft Word for quickly and easily creating documents such as letters, labels and envelopes. In Genus, the merge effect takes the standard mail merge and advances it many steps ahead, turning it into a full-fledged reporting solution that allows you to generate even more complex documents such as reports, catalogs, inventories, and invoices.

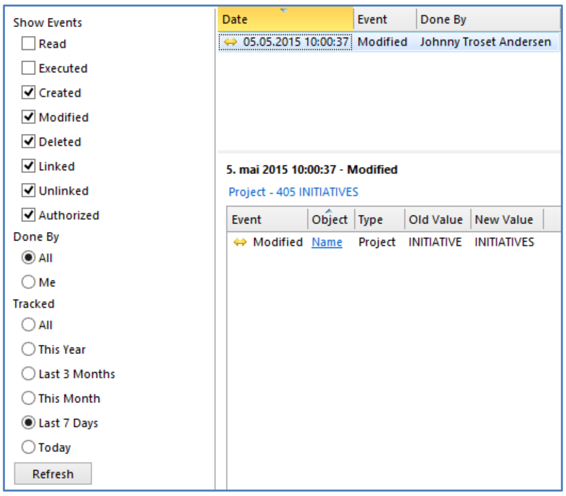
The output of the mail merge operation is not confined to the Microsoft Word Document format. You can save the merged document to many file formats, such as Adobe Portable Document Format (PDF), HTML, Microsoft Word (DOC, DOCX, DOCM), Open Document Text (ODT), Plain Text (TXT), Rich Text Format (RTF), and XML Paper Specification (XPS). Note that there is *not* any third-party software involved in producing these file formats, such as Acrobat Writer for PDF files.

The merge effect also allows you to merge data in Excel workbooks like the mail merge feature in Microsoft Word. Specifically, the merge effect allows you to automatically copy data into your Excel workbooks while retaining the layout, formatting and formulas in your workbooks. A variety of file formats are supported like PDF, HTML and Microsoft Excel (XLS, XLSX, XLSM, XLTX, XLTM, XLSB, XML) etc.

The merge effect is a built-in feature of Genus, and does not require Microsoft Office installed. It will run both on your desktop clients and on your Windows servers.

1.3.20 Event history

The Event History is a feature of Genus, which allows you to set up and view a chronological sequence of audit trails for objects, each of which contains evidence directly pertaining to and resulting from an event, like creating, modifying or deleting objects, or running dynamic content like an analysis, an agent or a task. The Event History enables you to reconstruct and examine the sequence of events and/or changes in an event for an object.



Show Events
□ Read
□ Executed
☑ Created
☑ Modified
☑ Deleted
☑ Linked
☑ Unlinked
☑ Authorized

Done By
● All
○ Me

Tracked
○ All
○ This Year
○ Last 3 Months
○ This Month
● Last 7 Days
○ Today

Date	Event	Done By
05.05.2015 10:00:37	Modified	Johnny Troset Andersen

5. mai 2015 10:00:37 - Modified

Project - 405 INITIATIVES

Event	Object	Type	Old Value	New Value
Modified	Name	Project	INITIATIVE	INITIATIVES

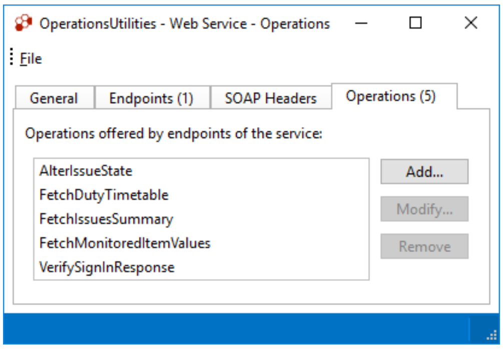
1.3.21 Agent

Agents are time scheduled actions modeled by using the effects mentioned above. Agents can be bound to a specific data set. Agents run automatically in the background on your server infrastructure.

1.3.22 Web services

A web service consists of operations (methods) exposed both internally (to a Genus client) and externally (to consumers on the network or the internet). The operations are made up of a sequence of actions with effects as described above. Genus supports consuming of web services exposed by other systems.

Genus supports both the SOAP specification and the REST architectural style. See section 3.1.8 for more information.



OperationsUtilities - Web Service - Operations

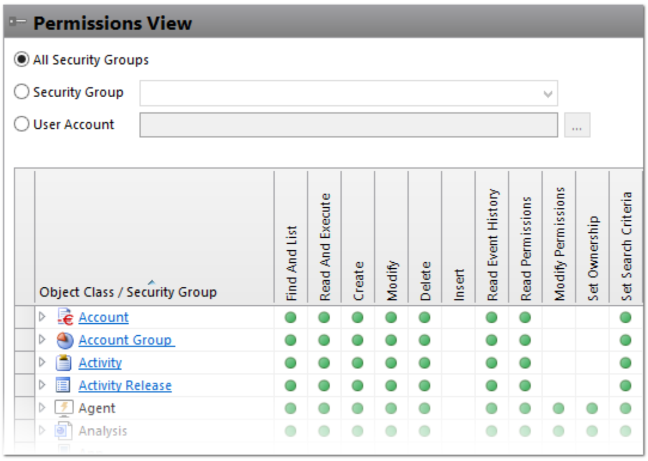
Operations offered by endpoints of the service:

- AlterIssueState
- FetchDutyTimetable
- FetchIssuesSummary
- FetchMonitoredItemValues
- VerifySignInResponse

1.4 Security and privacy

Genus uses a user or computer account to authenticate the identity of the user or computer, and to authorize or deny access to app resources (data and functions). The account establishes an identity for the user. The identity is then used to authenticate the user and to grant the user authorization to access specific app resources.

You collect users, computers and other groups into a security group, and then assign appropriate privileges and permissions to the security group. When you add a user to an existing group, the user automatically gains the privileges and permissions already assigned to that group. You can nest groups, that is, you can add a group as a member of another group, in effect adding permissions by inheritance.



Permissions View

● All Security Groups
○ Security Group
○ User Account

Object Class / Security Group	Find And List	Read And Execute	Create	Modify	Delete	Insert	Read Event History	Read Permissions	Modify Permissions	Set Ownership	Set Search Criteria
Account	●	●	●	●	●		●	●			●
Account Group	●	●	●	●	●		●	●			●
Activity	●	●	●	●	●		●	●			●
Activity Release	●	●	●	●	●		●	●			●
Agent	●	●	●	●	●		●	●	●	●	●
Analysis	●	●	●	●	●		●	●	●	●	●

The security model is based on Microsoft standards and is integrated with MS Active Directory for single sign on authentication. It is also possible to define security rules which enable automated permission modification based on business processes. You may choose to administer security accounts and groups within Genus or Active Directory.

The definition of objects in the app includes a security set up, which defines if objects of a certain type have class security (all objects), instance security (individual objects), or both. Class permissions are granted on app level and instance permissions are granted in the property dialogs of individual objects. You may also grant permissions based on conditions. You may define security rules which enables automated permission modification based on business processes.

Access to functions such as user management, user interface configuration, business rules management, etc. are granted with privileges given to security groups.

Access to Data Sets (e.g. the data for a single country when several countries share the same app model) is granted to security groups.

Genus supports ID-porten as Identity Provider on web. ID-porten is Difi's (Norway's Agency for Public Management and eGovernment) joint log-in solution for access to other public agencies within Norway.

Genus supports Trusted Zone Authentication for scenarios where Genus Services are protected within a trusted zone, where all incoming HTTP(S) requests are routed through a reverse proxy server that handles the user authentication.

1.5 Internationalization and languages

Genus supports multiple languages and both Norwegian and English (US) as default. Translation to other languages may be done by yourself or a third party.

The language support applies both to text (labels) which is a part of the Genus core (like Åpne / Öppna / Open) and text which is specific to your specific app (like Vikar / Vikarie / Temporary within the Staffing Industry in Norwegian, Swedish and English, respectively).

There is no built-in support for data content translation (i.e. translation of user entered data, like descriptions etc.), but web service calls may be added to utilize automatic translation tools like Google Translate or similar.

There is no need for separating data for different languages in different databases or schemas. However, in many cases a separation may be the correct solution with regards to performance, maintenance, sorting order variations etc.

When it comes to time and date formats, Genus adheres to the Regional setup on the client or server computer. Time data is currently not stored with time zone information, but this will be added based on customer demand.

Genus does not currently support Right-to-Left languages, but this will be added based on customer demand.

1.6 Best practices

The concepts described above make it possible to create a very wide range of applications across different industries. If a concept seems to be missing, you will probably find that the required functionality may be built by composing several concepts into the needed functionality.

1.6.1 Action orchestration

When building your app, and specially composing tasks out of actions and effects, there are many possible ways to achieve the same functionality.

We call this process action orchestration. Action Orchestration is used in Tasks, Agents, Web Services and Rules to define a sequence of actions that should be performed during execution. Specific tasks are performed within an action by adding effects, and to control the execution flow, each action is executed based on a condition.

During a project, Genus or one of its consultancy partners will help you out with action orchestration best practices.

1.6.2 Documents

A document is a property in Genus the same way a name or a birth date etc. Documents can therefore be distributed throughout the model connected to the appropriate business objects according to needs.

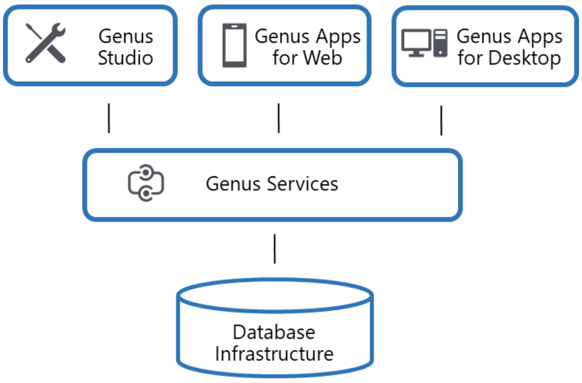
Your documents are stored in Binary Large Objects (BLOBs) in the database and are searchable as described above. Your document is automatically locked when a user opens it for editing, thereby supporting multi user handling.

You may extend your old apps by adding document functionality using Genus. You may even build your own Document Management System using Genus.

2 Architectural perspective

The architectural perspective is concerned with the technology as seen from a technical architect's point of view.

2.1 Layered architecture



Genus Services manages connectivity to the database layer and offer services to the Genus clients, i.e. Genus Studio, Genus Apps for Web and Genus Apps for Desktop. Modeled services interpret your app model at run time.

Genus Studio is the Microsoft Windows desktop client used by modelers for defining your Genus App model. The app model is stored in a database of your choice on the database infrastructure layer.

Genus Apps for Desktop is the end user client for Microsoft Windows interpreting your app model at run time.

Genus Apps for Web is the end user client for web browsers (on mobile phones, tablets and desktops), also interpreting your app model at run time.

The app model interpretation does not result in any performance penalty. Rather, knowledge about data (the app model) is used as input in our optimizers, resulting in improved performance compared to traditional programmed or packaged applications.

Genus Studio, Genus Apps for Web, and Genus Apps for Desktop communicates with Genus Services using the protocols http or https on ports 80 or 443.

2.1.3 Genus Apps for Web

Our web client, Genus Apps for Web, runs on the latest and second latest versions of all major browsers[3].

Support for browsers with a market penetration of 5% or less is generally removed in the next release, which implies that our support for Internet Explorer 11 and older versions may be removed in a future release.

Our web client may also be run in a web browser on a mobile or tablet device.

The web client runtime is itself built using Html5, JavaScript and CSS (Cascading Style Sheets). Modelers do not have to think about Html5, JavaScript and CSS. The modeling is done inside Genus Studio as before, using the same modeling primitives as for Genus Apps for Desktop.

Genus Apps for Web requires an installation on an application server, which could be the same server that hosts the Genus Services installation.

Caching takes place both on the client and services layers.

The communication between our web client and Genus Services is RESTful using JSON and all modeled security is honored.

2.1.4 Genus Apps for Desktop

Our desktop client runs on 64-bits editions of Windows 10, Windows 8.1, Windows 8, Windows 7, Windows Server 2016, Windows Server 2012 R2, Windows Server 2012, and Windows Server 2008 R2

[3] Mozilla Firefox, Google Chrome, Windows Internet Explorer 11 and Edge, Apple Safari (Mac OS X, iOS 9.x and later). Other browsers may be supported on customer demand.

SP1, all with .NET Framework 4.6.1 or later. See https://docs.genus.no/developers/installation-and-configuration/system-requirements.html for more details.

2.1.5 Genus Studio

Genus Studio runs on the same operating system versions as Genus Apps for Desktop.

2.1.6 Genus Services

Genus Services runs on Virtual Machines, either available on-premises at the customer's location, at a third-party hosting service provider, in Microsoft Azure or any other cloud provider supporting IaaS (Infrastructure as a Service). The term *Genus Server* is often used for a virtual machine running one or more Genus Services.

Genus Services will in a future release be made available on the PaaS (Platform as a Service) layer of Microsoft Azure. In preparation for the transition, Genus already supports administration of services by defining nodes (i.e. servers) and logical grouping of nodes into groups.

A node can be configured to run one or more services. App model objects such as data marts, web services, and agents, are deployed to a node group. Each member of a node group is connected to a node and a data set. Objects (i.e. services) deployed to a node group is processed by any node within that group which provides the requested service for a given data set. A node is currently to be interpreted as a server (e.g. a virtual machine), but is in a future PaaS release to be interpreted as nodes in a microservices based cluster. Other cloud providers than Microsoft Azure will be considered supported on the PaaS layer based on customer demand.

The long-term strategy is to make Genus Services available as an aPaaS, i.e. as an *Application* Platform as a Service, built on a microservices architecture.

Supported server operating systems are Windows Server 2016, 2012 R2, 2012, and 2008 R2 SP1, all with .NET Framework 4.6.1 or later. Only 64-bit editions are supported. Server Core Role is supported for Windows Server 2016. See https://docs.genus.no/developers/installation-and-configuration/system-requirements.html for more details.

Genus Services manages connectivity to the database layer, and communicates with the Relational Data Base Management System (RDBMS) using OLE DB Providers and native .NET Data Providers for e.g. Microsoft SQL Server, Azure SQL DB, Oracle, DB2, MySQL and PostgreSQL. Other .NET native data providers will be added based on customer demand. See https://docs.genus.no/developers/installation-and-configuration/system-requirements.html for more details.

Genus Services utilizes database statistics for optimizing database queries. We therefore recommend verifying your application's database maintenance procedures, making sure that the database statistics are kept regularly up to date.

Support for non-relational databases (NoSQL) will be added based on customer demand. Genus naturally lends itself to NoSQL databases, since data internally in Genus is handled like objects, as opposed to relational structures.

Genus Services offers several services, primarily to be utilized by the Genus clients. Some of them are however available externally.

The services available both to the Genus clients and external consumers are:

- **Authentication service**:
    - Provides logon authentication and user authorization. Your users are granted access to your Genus app (both data and functions) according to their credentials, based on Microsoft Active Directory principles with some extensions. The authentication service is integrated with your Active Directory, providing the Genus clients and your users with single-sign-on (SSO).

  - You may alternatively configure Genus without Active Directory and towards your own user database, requiring your users to enter their user name and password during logon.

  - Genus also supports *ID-porten* as Identity Provider for Genus Apps (web edition). *ID-porten* is Difi's (Norway's Agency for Public Management and eGovernment) joint log-in solution for access to other public agencies within Norway.

  - Genus supports Trusted Zone Authentication for scenarios where Genus Services are protected within a trusted zone where all incoming HTTP(S) requests are routed through a reverse proxy server that handles the user authentication. The reverse proxy server will add a special HTTP header whose value is the identity of the authenticated user.

  - Genus supports the open authorization standard OAuth2, currently available in beta for selected customer projects.

- **Web services**: Responsible for exposing the web services defined in your app model, to be consumed by external partners, or also internally by the Genus clients.

The services available internally both to our web and desktop clients are:

- **Action service**: Responsible for executing scheduled business actions as described by you in your app model. One example would be distributing result reports to your corporate management the first day of every month.

- **Mail service**: Responsible for interfacing with your mail services by SMTP for distributing email.

- **App model deployment service**: Responsible for caching and deployment of your app model towards the Genus clients. The app model is stored in a database of your choice. The App Model Deployment Service caches the model in an internal compressed and optimized format, offered to the Genus clients. This internal format is not externally available.

- **Core service**: Responsible for interpreting your app model at the server tier. The same core also runs on Genus Apps for Desktop. Genus Apps for Web runs a scaled down version.

The services available internally only to Genus Apps for Desktop are:

- **Transaction service**: Responsible for retrieving, updating, creating and deleting objects on your database server. The provided services are executed in a transactional context, supporting data consistency. An audit trail is created for changes to objects, allowing the object history to be viewed or searched. The transaction service is utilized by the Genus clients, but also the other services.

- **Database schema service**: Responsible for interpreting RDBMS tables. This restricted service is only used by your users trusted with the right to edit or change your app model, i.e. Genus Studio users (app developers).

The services available internally only to Genus Apps for Web are:

- **App services**: To keep our web client run time interpreter as small and effective and secure as possible, model interpretation is divided between the client and the app services. Genus Services offers specialized services for handling this division. The services are:

  - Read Meta Data: Reads the app model from the server. The app model is cached on the server.

  - Read Data: Reads data from the server upon requests from the upper web client layers.

  - Request Data Change: Our web client is in itself not allowed to decide data changes, but must ask Genus Services to perform the change. The request may fail, in which case the web client must handle the situation accordingly.

  - Execute Action: Execute actions, like execute a task when clicking a button.

2.2 Scalability

All services are multi-threaded, thereby allowing you to scale your Genus Services vertically (scale up), i.e. utilize more memory and CPUs on your server or node.

All services are stateless, providing resilience at the Genus Services tier, meaning that several Genus Services can be clustered using Microsoft Network Load Balancing Technology (or similar), allowing you to scale horizontally (scale out), i.e. adding more servers or nodes to your Genus app.

Genus contains the ability to divert read only transactions to separate and replicated database servers, allowing to scale horizontally also on the database server layer. Support for NoSQL databases, which inherently contains the ability to scale out also for create, update and delete transactions, will be supported based on customer demand.

2.3 Concurrency

Concurrent users are defined as the total number of people using a resource within a predefined period of time. We usually use a time period of one minute.

A typical commodity infrastructure is:

- One virtual machine for Genus Services with 2 – 6 CPU cores, 4 – 8 GB RAM, 50 GB of disk space.

- One database server with 2 – 4 CPUs or cores, 6 – 12 GB RAM, disk space dependent of amount of data required.

Such an infrastructure may handle up to 250 concurrent Genus Apps for Desktop users and 100-200 concurrent Genus Apps for Web users.

If more users log into the system without performing an action within the one-minute period, typically one could double or triple our number of concurrent users. Most of our customers are using virtual servers, with a greater flexibility to adjust allocated resources dynamically. The numbers above are highly dependent of the app’s functionality, but are provided to give you an idea of a typical infrastructure.

Genus currently runs in higher configured production environments with up to 8 000 registered desktop users (approx. 1 500 concurrent users). More users could run concurrently by adding more Genus Services nodes.

2.4 High availability

Since Genus Services are stateless, a failure in a single Genus Services in a cluster environment will not affect the entire app. However, there is not built in support for *automatic* failover or shutdown of failed servers or nodes. Such failover capabilities will be added in a future release as part of our transition to a microservices architecture. Our apps are currently meeting the availability criteria required by our customers.

A setup with Microsoft Network Load Balancing Services (NLBS) or similar, allows for manual failover, i.e. when a failing Genus Services is detected, this server may be removed from the cluster. Such a load balancing setup is currently in use by some of our customers. These customers have experienced an uptime of 100% the last year (minus planned downtime).

3 Integrational perspective

The integrational perspective is concerned with the technology as seen from a system integrator's point of view.

3.1 Integrations at the client tier

3.1.7 Genus Apps for Web

Our web client supports actions like send email, dialing, input-adjusted keyboards, camera, drag-and-drop, and mailto and url links.

Genus Apps for Web has built-in support for reporting page hits and events to Google Analytics.

3.1.8 Genus Apps for Desktop

At the client tier, Genus is tightly integrated with functions in Microsoft Office, providing mail merge, copy/paste, drag & drop, export/import of files, sending of e-mail and objects (xml, iCal, vCard, File, e-mail).

Our desktop client utilizes the authentication service provided by Genus Services in order to provide single-sign-on (SSO) for your end users.

Our desktop client utilizes its own protocol handler at the client tier, allowing users to send email containing shortcuts (links) to your app objects to each other, making it possible for the user receiving the link to click on it to automatically open the desktop client to access and edit the linked object. Shortcuts can also be placed on the desktop or in your file system.

3.2 Integrations at the Genus Services tier

Genus supports a service oriented architecture (SOA) by exposing and consuming Web Services on the Genus Services tier, supporting both the SOAP specification and the REST architectural style.

REST Web Services

Genus provides two kinds of RESTful APIs: *Genus Modeled REST-services* and *Genus Graph*.

Genus Modeled REST-services provides light-weight, data-centric access to resources in the app model. A resource is an abstract concept of "something" accessible by a given URL. A resource may support some or all CRUD[4]-operations, using standard HTTP-verbs. To address the resources a service contains one or more resource paths. Each resource path defines a directory like human readable URI, such as http://my-service/persons/{person-id}. The job of a resource path is to identify a resource or a collection of resources. The operations which is supported for a given resource path is defined by its resource methods. Each resource method implements a HTTP-verb, such as GET, PUT, DELETE, or POST. That is, the same URI defined by the resource path can be called with different HTTP verbs to perform different operations. Data in the request message, such as headers, query parameters, and body, are collected in data sources defined in the resource method. The actual operation is performed by executing actions. Resources contained in the request or response message body can be represented on JSON or XML format.

The RESTful services may be exposed using the OpenAPI Specification (Swagger). The specification may be represented in JSON or YAML, while request/response data elements may be in JSON or XML. Content of type "multipart/form-data" when consuming REST services is supported.

Genus Graph is a representation of your app model adhering to the OData 4.0 standard, and provides a structured and well-known API into your model data. Genus Graph allows rich querying capabilities including filtering, sorting and formatting, and can support JSON- or XML-based exchange formats. Metadata is automatically generated and provided according to the OData specification, based on which parts of your app model you choose to expose to clients.

[4] Create, Read, Update, Delete

SOAP Web Services

Web Services may be exposed and consumed using the Web Service Description Language (WSDL) for providing the web service metadata.

Genus currently supports the SOAP 1.1 and 1.2 protocols, utilizing Windows Communication Foundation (WCF) and the Microsoft .NET Framework environment.

The SOAP standard offers a wide selection of assorted styles and flavors when it comes to the actual encoding of the SOAP message payloads. When a web service is published by Genus, a WSDL supporting the "document/literal wrapped" binding style is generated. The "document/literal wrapped" binding style is also preferred when consuming an external Web Service. However, for legacy reasons, the "RPC/literal" style is also supported. Genus supports publishing of endpoints accepting POX/JSON payloads.

XML Schemas provide a means for defining the structure, content and semantics of XML documents, and consists of a set of components such as type definitions and element declarations. In Genus, you define a XML Schema by building a structure using declarative components such as groups and fields which then is automatically translated to XML when needed.

Other integration techniques

Genus also support integrations using classic techniques as file import and export on the Genus Services tier. However, Genus recommends building new integrations using web services technology.

Our customers have implemented many integrations with third party products using our technology. Examples of third party products are Microsoft Dynamics AX (former Axapta), PeopleSoft Financials, Oracle Fusion Middleware (Oracle BPEL Process Manager), Norwegian Central Securities Depository (Norwegian: Verdipapirsentralen), Agresso (Norwegian ERP product) and Visma (Norwegian ERP and CRM product).

You may define calendars in your app model, like customer meetings from your Genus CRM app or events from your Genus social app, and share them with others by a URI. Your users may subscribe to calendars by adding a subscription in their mobile device, in Microsoft Outlook, or similar. Calendar sharing is based on the Webcal scheme for accessing iCalendar files. Calendars are shared read-only, but each shared item contains a link, so your user can open your Genus app and inspect or edit the calendar item. Our calendar sharing is *not* based on WebDAV.

You may also define contacts in your app model, like customer contact persons from your Genus CRM app, and share them with others by a URI. Your users may subscribe to contacts by adding a subscription in their mobile device. Contact sharing is based on a read-only CardDAV server.

3.3 Integrations at the RDBMS server tier

Genus Services uses standard SQL (Structured Query Language) to access your databases, and does not make use of stored procedures or any form of logic on the RDBMS tier.

Genus does not recommend integrating with other apps directly on the RDBMS tier, since this bypass the rules of your app model. On the other hand, Genus does not prevent you either, making this an integration alternative in some cases, like running Genus towards a replicated copy of your production database for reporting (read only) purposes.

3.4 Integrations by programming is not supported by design

If specific functions currently not provided by Genus are needed, several mechanisms are supported to implement these functions tightly integrated using actions such as “Consume a REST Service”, “Consume a Web Service” and "Run a Program".

Since Genus is a true model driven app development tool, it is not possible to insert program code into Genus. This represents a risk that Genus may not allow a specific function or facility to be accomplished, and it is not possible to extend the functionality of Genus without Genus performing these changes to the tool itself.

However, Genus has been marketed since 1996, and is used by many large Norwegian and international organizations in different industries. The technology is therefore mature and feature rich, reducing the possibility for lack of functionality to a minimum. Also, any missing functionality is added quickly to the tool based on customer demand.

4 Implementational perspective

The implementational perspective is concerned with the technology as seen from a project leader's point of view.

4.1 Deployment

The deployment of Genus is made up of the following components:

1. The deployment of our desktop client on your Microsoft Windows clients, such as PCs or terminal servers. The installation is performed using standard Windows Installer technology. The client can be installed using an internet address (URL) or an otherwise provided setup file. Also, in corporate environments our desktop client may be preinstalled by your system administrator. You can configure the client to automatically update itself, similar to Microsoft's Windows Update concept.

2. The deployment of Genus Apps for Web on your web server(s), for executing web apps in your users' browsers. The installation is performed using standard Windows Installer technology.

3. The installation of Genus Services on your app servers. The installation is performed using standard Windows Installer technology. In addition, you need to configure your Genus Services using Genus Services Configuration tool on your server.

4. The deployment of your app model to Genus Apps for Web, Genus Apps for Desktop, and Genus Services. Your app model is stored in a database on your RDMBS server, and a cached and compressed copy is held in memory by the app model deployment service at the server tier. Changes to your app model is automatically detected by all components in Genus Apps, like Microsoft's Windows Update concept. During deployment, the users are warned before forcing a restart. When deploying, you specify the desired time of day, in effect making it possible to deploy changes to your app model to a time outside your business hours.

Genus Apps may be hosted on your own servers, by a third party or in the cloud.

4.2 Release stages

A software release is a distribution of an upgraded version of a software product.

Genus Apps, as all other software, goes through distinct stages that describe the stability of the software and the amount of development or testing it requires before the final release of the upgrade.

Genus may be distributed in one of the following release stages:

- **Pre-alpha.** This version is not feature complete, and typically used by the developers prior to software testing. The activities performed on such a version can include requirements analysis, software design, software development and unit testing. The version is not distributed outside the development environment, but described here for the reason of completeness.

- **Alpha.** The version delivered to the software testers, i.e. persons different from the software developers. The version is not normally distributed outside the development and test environments.

- **Beta.** The first version released to outside testers or for evaluation purposes. The version generally includes all features, but may also include known issues and bugs of less serious variety. The software is not yet ready for release, and new source code will be added to this version.

- **Release candidate.** This version has the potential to be the final product upgrade. No new source code will be added to this version, but it may be bug-fixed (patched).

- **Release.** The final version of the upgrade with a quality suitable for wide distribution and use by end-users.

When producing a Release candidate, a copy (branch) of the source code is created. It is this copy which later is used to produce a final Release, or it may be bug-fixed (patched) to produce a new Release candidate. The preceding stages (Pre-alpha, Alpha and Beta) may not be patched.

The Alpha, Beta and subsequent stages have built-in mechanisms easing the analysis of technical support incidents reported by testers or end-users. This does not apply to the Pre-alpha stage.

As a Genus customer, you are required to upgrade the Genus Apps tool at least yearly. New versions are released at least three times a year. It is for you to decide which release to use for your yearly upgrade. Note that it must be the newest release, and not the second newest release or older.

Internally, we use Git as our source code revision control system and adhere to the practice of continuous integration (CI).

4.3 App modifications

Your selected and trained users may make modifications to your Genus app for certain functionality like:

- **Table views.** Modify sorting, columns etc. to suit the user's needs.
- **Mail merges.** Selected users may modify Microsoft Word or Excel templates.
- **Reports.** Selected users can modify facts, formulas, axis dimensions etc. on existing reports, or create new ones.
- **Analysis and data marts.** End users may use Analysis to visualize data defined in Data Marts and modify the visualization. Data Marts may be subdomains or parts of your app model.
- **Searches.** Selected users may define and save searches to be used by your end users.

The above modifications are done directly in your app (using our desktop client). Authorization for modifying is controlled by a modeled security setup. Most of our customers choose to make such modifications themselves.

Most Genus customers outsource the modifications of all other app logic to our consultancy partners. However, you may also choose to do them yourself, which requires proper Genus Studio training.

In addition, there is a small Genus Services configuration tool (Genus Services Configuration) supporting the bootstrap setup of the app, plus supporting certain highly technical configuration options. Most of our customers outsource such configuration modifications to our consultancy partners.

4.4 Extract, transform and load (ETL)

Genus supports importing and exporting of data primarily for needs close to the end user, such as importing customer contact persons from an XML file or exporting report data to a flat comma separated file.

Genus is *not* focused towards supporting extracting, transforming and loading of substantial amounts of data in data warehousing. This is left to specialized ETL software products.

4.5 Error handling

Errors are handled differently depending on the nature of the error.

App model errors (e.g. tasks failing) are, after being routed though customer 1st or 2nd line support, being handled by our partner's app model responsible team. The audit trail is investigated (history on the data objects, or execution log from the failing task), and the error is recreated in a non-production environment. Fixes to such errors may be deployed directly in production with minimal downtime (1 second downtime with user alert prior to deploy), and are typically modeling errors.

When such errors are analyzed by the app responsible team, a utility called Genus Log is utilized. This is a utility for monitoring all effects and steps when using any Genus app, and the level of detail in this log can be adjusted to the level of scrutiny necessary.

Genus errors (i.e. errors in the Genus Apps tool itself) during run-time are reported by email (through customer 1st or 2nd line support) to our partner's app model responsible team. Genus Apps for Desktop has built-in functionality for fetching such errors, opening an email window with files (log, screenshot, and system info and bug report) attached.

Server side errors, such as web service operations failing, are possible to track by the audit trail of the web services.

Unexpected events (server side error events, consistency constraints, concurrency errors, communication errors, run-time crash etc.) are logged in the Genus Services event log, and all these events may be viewed in the Windows Event Viewer on the server or the client. Logs may be shipped to our partner's app responsible team, or the team is granted access directly to the server(s) for investigation. The Genus Services event log is an important source for tracing both web and desktop client errors.

In addition, all run-time crashes are automatically captured using Sentry. If you do not want such capture, you may turn it off in your Genus Services configuration. You may install a Sentry server on premises, if you do not want to send stack traces etc. to the online Sentry services.

4.6 Transactional throughput characteristics

We operate with a mix of Key Performance Indicators (KPI) and guidelines to control the overall performance. These KPIs and guidelines are checked during system tests to ensure that our customer's requirements are met. The KPI characteristics and guidelines are described below.

4.6.1 Web service operations per second

A single Genus Services instance with a typical production configuration is able to handle about 2 web service operations per second. This may not seem very impressive, but note that web service operations are not comparable to traditional database transactions.

One web service operation may result in 1 to e.g. 100 database transactions (including reads); this depends entirely of the definition of the web service. For example, in the context of Web Time entry, the most performance intensive web service operation is the timesheet *update* operation, since up to a whole month of reported time rows may be updated in one operation. In comparison, for a mobile version of a Web Time entry, the number of operations per second would be much higher (>10) since updates normally are performed on a "per day" basis.

The number of Web Service operations per second is scalable by adding more Genus Services, in addition to tuning the action orchestration of your web services.

Database read transactions

The guideline used in our performance analysis during system tests, is that no single database read transaction should take more than about 0.2 seconds, to take full advantage of the database systems preference for a high number of small and simple transactions over large and complex transactions. Most transactions lasting longer than this limit, are carefully inspected and rewritten, optimized or divided in smaller transactions, but carefully weighed against latency considerations. Some transactions can last longer, if they are infrequent or do not impose a performance risk to the system as a whole.

The simple guideline above has proven sufficient for all our app models. In addition, Genus contains the ability to divert *read only* transactions to separate or replicated data sources. The maximum database read transaction performance is therefore very high. However, currently none of our customers is in need of using this ability.

For large apps, we estimate the number of read transactions per second and test the throughput for key transactions in our system tests.

Database modify (including insert and delete) transactions

In our experience, more than 95 % of the transactions in our apps are read transactions, and less than 5 % are modify transactions. In general, it is important to optimize for read transactions. However, in large apps the number of modify transactions is substantial and must also be considered and handled carefully.

Our main guideline is that no single database modify transaction should take more than about 0.5 seconds. As with read transactions, database systems prefer a high number of small and simple modify transactions over large and complex ones. However, modify transactions are more difficult to simplify and divide, so our set of transaction design guidelines are more complex. Here are the most important ones for relational databases:

- Spread the load on different database tables. If necessary, split the hotspot database table *horizontally* using database mechanisms like Oracle materialized tables or MS SQL Server indexed views. Consider splitting the hotspot database table *vertically*, by moving columns out in other tables.

- Review the table indexes carefully. Make sure table layout-deciding indexes match well with your modify transaction characteristics. Be careful not to add too many read indexes, since this may slow updates. For insert transactions, the main index should spread the insert load to various parts of the table, to avoid I/O hotspots within a table. Similar considerations apply for update transactions.

- Avoid using modify transactions containing more than one update, delete or insert, or combinations thereof. If possible, split these to single transactions and rather build in error handling to take care of any inconsistencies.

For large apps, we estimate the number of modify transactions per second and test the throughput for key transactions in our system tests.

4.6.2 In-memory databases

Support for in-memory databases will be added based on customer demand. For data mart, discovery and analysis workloads, Genus Services uses its own, highly optimized in-memory database.

4.7 Network requirements

4.7.1 Network bandwidth

The numbers below are approximate numbers at its best, and we cannot guarantee their correctness. If network bandwidth is a concern, we recommend measuring the bandwidth usage in a test environment before full scale deployment of the specific customer app.

The Genus Apps for Web client

Our web client is built for handling EDGE bandwidth or later (3G - UMTS, 3.5G - HSPA, 4G - WiMAX/LTE) data transmission rates. A typical actual bandwidth range for EDGE is 120 to 384 Kbps.

The Genus Apps for Desktop client

The network bandwidth requirement is measured to an average of about 30 Kbps per active user, i.e. users working actively with a desktop client. This however is highly dependent of the customer app's functionality. If an app has a high rate of picture and document traffic, the bandwidth requirement will be higher.

1 Mbit/s should handle about 30 to 40 active users, given a requirement of 30 Kbps per active user.

4.7.2 Network latency

Generally, if network latency (network round trip time) is a concern, Genus recommends measuring the network latency in a test environment before full scale deployment of the specific customer app.

The Genus Apps for Web client

Our web client is built for handling EDGE or later latency rates. Latency for EDGE seem to vary a lot, typically from 150 to 600 milliseconds. Anyhow, the web client to to Genus Services response timeout is set to 15 seconds, to handle bad network conditions.

The Genus Apps for Desktop client

A function that requires many round-trips from the desktop client to Genus Services will be vulnerable for high latency. Vulnerable functions in Genus will typically be opening of detailed objects (typically in Forms or Tables) with many rules and references to other objects, but also reports with many value calculations. Opening of a complex Form may require 5 to 20 round-trips. A user's "limit of patience" for frequently executed functionality is about 1 second.

We know that our desktop client works excellent and with good performance on a latency of 15 to 20 milliseconds (https, between the cities Oslo to Bergen in Norway).

We also know that our desktop client works satisfactory on a latency of 50 - 70 milliseconds (https, between Oslo, Norway and the Microsoft Azure data center in Amsterdam, The Netherlands), without adjusting application functionality (by remodeling) compared to a lower-latency environment. However, somewhat higher user-experienced delays are to be expected, but should be within a user's "limit of patience".

At environments with a higher latency of 70 milliseconds you may have to adjust (by remodeling) the latency-vulnerable functions of your Genus Apps for Desktop application. This is highly dependent of the functionality of your application.

At even higher latency environments you may expect more parts of your Genus Apps for Desktop application to be adjusted. However, if you experience a latency above 100 milliseconds we recommend you to raise an issue with your network vendor or use a data center closer to your desktop clients' physical location.

Within a continent (Europe, North America etc.) a latency of more than 100 milliseconds is not to be expected. See http://www.azurespeed.com/ and https://wondernetwork.com/pings for latency examples between data centers and cities around the world. Note that in our experience the application latency is 10 – 30 % higher than the numbers given in the examples.

Bibliography

[1] Qlik, "What makes QlikView unique," January 2014.