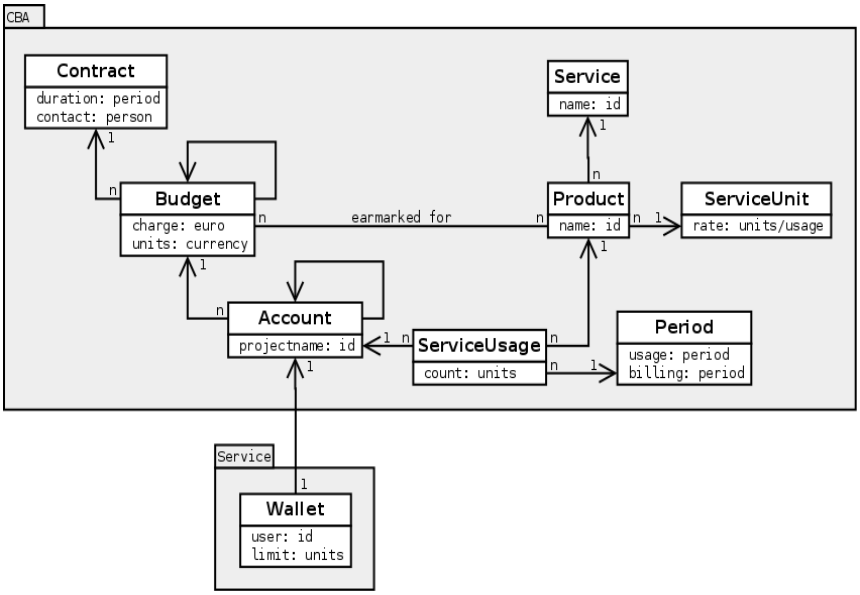
Accounting for SURF ResearchCloud related services

The services provided by SURF are used both directly by researchers as well as indirectly, via IT staff functions in the organizations. Institutes need to be able to associate the costs of indirect usage of data processing services provided by or via SURF with appropriate charge-back to internal cost centers of the beneficial research projects. Ideally we automate this process where feasible so that accounting can be performed efficiently.
This document investigates 1) how SURF accounts for the usage of its services and 2) how institutes can build on SURF's accounting reports to distribute costs appropriately across its costs centers. Our use case involves SURF ResearchCloud as a service and Utrecht University as an academic organization yet we expect that the processes, policies and issues will apply to other services and academic organizations as well.

How SURF accounts for the usage of its services – concepts

(emphasized words refer to concepts used by SURF in accounting reports)

Service usage preparation:
A **Contract** is agreed between SURF and Utrecht University for provisioning of services up to an agreed amount of money for the duration of a period of time. The Contract may also stem from research grants and include a funder as a party. The Contract is detailed out in at least one **Budget**, an allocated part of the funds of the Contract that has a list of SURF Products on which the Budget may be spent and a selected unit of accounting, for instance compute hours.
A Budget can be subdivided into **SubBudgets** to earmark and allocate parts of the Budget for a purpose.



Debit transactions for the use of products are registered on an Account. In turn, an Account draws from a related (Sub)Budget which balances the debit transactions with available credit. The association between an Account and its Budget may change over time, for instance when a new Budget is created each annum. SubAccounts can optionally be defined underneath the main Account to represent different activities that draw from the same budget.

When researchers use a SURF **Service**, they must upfront specify the Account (in the service), sometimes referred to as a "project" (e.g. HPCCloud), that is used for any charges by the **Products** that comprise the SURF Service. To prevent unpleasant surprises caused by excessive charges, the Account exposure can be limited by authorizing individual users to spend up to a maximum amount of units on a SURF Product. The limited amount is registered as a **Wallet**, which is linked to a similarly-named SubAccount. Note that currently only SURF ResearchCloud supports this granular level of cost control where users specify a Wallet instead of an Account.

The actual billing of resource usage:

Each accounting line specifies a charge to an account and states the priced item (**Product),** a rated unit of measure for that item (**ServiceUnit**) and the actual number of units of that item that have been used (**ServiceUsage**). See appendix A for an example accounting spreadsheet report.
Information-only lines are added by using a Product with a 0 cost rate per unit, for example to list a human-readable amount of disk usage in terrabytes. The actual cost per unit used for an item is usually implicit in the current accounting reports, yet can be derived in two steps as:

rate-in-credits = **Usage / ServiceUsage**
rate-in-euro = rate-in-credits * (C**harge / Budget**)

The use of SURF services is listed as one or more accounting lines per **Period** (calendar month). Currently the Period covered is only associated with actual use. As a result, accounting reports may show different figures for a particular month due to corrections on transactions. A planned version of the accounting system will also associate transactions with billing periods. Corrections and other yet unaccounted transactions that emerge after their month of consumption will then be billed to a later month.

Utrecht University accounting and charge-back related policies

Transparency on research project related expenditures is required by many stakeholders. The stakeholders may include external funding agencies such as NWO and other consortium partners (public sector) as well as any companies (private sector) that co-fund a project.
For financial reporting reasons, it is crucial that any SURF service charges can be matched with the relevant research project. The evidence that accompanies an invoice must allow the researcher and other stakeholders to assess that the invoice is *correct, timely and complete*. In addition, external funding agencies may require evidence that an amount of service has been *consumed,* for instance compute-hours. In general, an invoice is correct if it can be matched unambiguously with prior agreed arrangements. Orders should include at least a service description, the associated once and/or recurring costs, due dates for payment and a reference to the project or cost center.

Note that in the case of indirect usage (service ordered via central IT function) the SURF invoice will initially be processed by the central IT department. Subsequently the IT department may internally charge-back the research project or decide to draw from a central cost center if charge-back is inefficient or otherwise not appropriate.
Interestingly, as can be deduced from the above policy, in some situations the selection of a cost center will be made *after* the costs are incurred.

How to link SURF accounting and Utrecht University accounting

To support financial reporting, the naming of SURF Budgets, Accounts and Wallets should be chosen carefully so that they can be used to identify the related Utrecht University research project and cost center.

SURF	UU envisioned usage	Projected UU concept
Budget	This concept is well-suited to relate to sources of funding along with planned annual amounts.	Limit order (budget linked to a ITS contract with SURF)
Account	This concept maps well to actual expenses related to a research project or part thereof.	Research project id (within UU this id is linked to a WBS cost center for optional charge-back purposes)
Wallet	This concept links actual expenses to the person who has ordered the service	Research-project id with solisid or researcher name added as suffix

UU's Research IT department agrees a general purpose contract (with limit order) with SURF, which represents a subscription to services. This results in a SURF budget.
The SURF budget is used to accommodate many UU research projects. UU provides each Research project with a unique short-name which is subsequently used as SURF Account and as such made known to SURF. Each UU research project member is provided with a Wallet linked to the Account (this needs to be registered as such at SURF). As SURF reports service usage per wallet and per account, expenditures can be traced back to both the research project serviced as well as the individual responsible for ordering the service.

We have created a proof-of-concept application to investigate if the current SURF accounting spreadsheet report can be machine-interpreted and used for charge-back purposes. Example output of our application is included in Appendix B.
The proof-of-concept application interprets the SURF service usage reports (spreadsheets), enriches these spreadsheets with cost center numbers per account, and produces a report that can be used to drive internal charge-back activities.

SURF ResearchCloud

SURF ResearchCloud uses the notion of a "Collaboration" to refer to a set of people authorized to use a service together. We can match the name of a collaboration with the short-name of a research project so that authorizations and accounting are aligned.

Appendix A: Example (partial) of a SURF accounting report (spreadsheet)

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	V
1	Code	Description	Valid_from	Valid_to	Account	Charges	Unit	Amount	Budget	Usage	Balance	%used	Status	Product	SrvUnit	SrvUsage	trend	2021-03
2	2010101_01	RCCS-credits 2020-2021	01-11-2020	31-10-2021		17,010	credit	600,000	600,000	182,126	417,874	30%	2-usage	*	credit	182,126		41,429
3	2010101_01/L1	RCCS-credits 2020-2021	01-11-2020	31-10-2021	UU		credit		158,115	0	158,115	0%	1-no-usage	*	credit	0		
4	2010101_01/L1/1	RCCS-credits 2020-2021 - subbudget for lisa-gpu-nikosk	01-11-2020	31-10-2021	lisa-gpu-nikosk		credit		100,000	358	99,642	0%	2-usage	*	credit	358		358
5	lisa-gpu-nikosk	/UU/lisa/lisa-gpu-nikosk			lisa-gpu-nikosk		credit			45				lisa:cpu	SBU	39		39
6	lisa-gpu-nikosk	/UU/lisa/lisa-gpu-nikosk			lisa-gpu-nikosk		credit			314				lisa:gpu	SBU	275		275
7	2010101_01/L1/2	RCCS-credits 2020-2021 - subbudget for lisa-uuuse	01-11-2020	31-10-2021	lisa-uuuse		credit		21,385	3,625	17,760	17%	2-usage	*	credit	3,625		

Row 2 of the spreadsheet report represents a Budget. This budget (its name is in column A) is divided into several SubBudgets (examples in row 3, 4 and 7). The subbudget of row 4 is partially consumed by the use of two products, as shown in rows 5 and 6. The name of a product is stated in column N. The related account is stated in column E.

Appendix B: Proof-of-concept application output

Our Python application (source in Appendix C) imports a sample SURF accounting spreadsheet using the Python Pandas module. It attempts to categorize each row to see if it pertains to a Budget or an Account. Unrecognized formats result in rejection of the spreadsheet.
Subsequently it links an Account to a known cost center. Currently, for demonstration purpose, the list of cost centers is held in a dictionary. Ideally this would be held in a database and filled using information received from researchers.

Sample program output using the spreadsheet partially listed in Appendix A as input:
$ python3 read-account.py
COST CENTER (ACCOUNT) : USAGE IN CREDITS
WBS-43788 (Waysdorf-tweets) : 613
WBS-50311 (lisa-gpu-nikosk) : 358

Conclusion:
The SURF report can be read programmatic and used for charge-back purposes.
There is a minor limitation: The report does not explicitly link a SubBudget to a parent Budget other than by name (a Budget appears to be the prefix of a SubBudget). This missing feature currently does not limit our automated interpretation for charge-back purposes, yet it might impact other applications such as budget-related analysis.

Appendix C: Proof-of-concept Python program

```
#!/usr/bin/python
# Author: Ton Smeele
import csv
import sys
import getopt
import re
import textwrap
import os
import collections
import pandas as pd
import warnings

C_TYPE = 'Type' # this column will be added to the dataframe to describe type of row
T_UNCLASSIFIED = 'Unclassified'
T_BUDGET = 'Budget'
T_ACCOUNT= 'Account'
T_TOTAL = 'Total'
T_ALLOC = 'Allocation'
C_CODE = 'Code'
C_DESCRIPTION = 'Description'
C_VALIDFROM = 'Valid_from'
C_VALIDTO = 'Valid_to'
C_ACCOUNT = 'Account'
C_CHARGES = 'Charges'
C_UNIT = 'Unit'
C_AMOUNT = 'Amount'
C_BUDGET = 'Budget'
C_USAGE = 'Usage'
C_BALANCE = 'Balance'
C_PERCUSED = '%used'
C_STATUS = 'Status'
C_PRODUCT = 'Product'
C_SRVUNIT = 'SrvUnit'
C_SRVUSAGE = 'SrvUsage'
C_TREND = 'trend'

# COLUMNS should list all the fixed columns of the source file (not columns that depict a year/month)
COLUMNS = [C_CODE, C_DESCRIPTION, C_VALIDFROM, C_VALIDTO, C_ACCOUNT,
C_CHARGES, C_UNIT,
      C_AMOUNT, C_BUDGET, C_USAGE, C_BALANCE, C_PERCUSED, C_STATUS,
C_PRODUCT, C_SRVUNIT,
      C_SRVUSAGE, C_TREND]


INPUTFILE = '2010101_01.20210906.xlsx'
```

```
ACCOUNT2WBS = {'lisa-gpu-nikosk' : 'WBS-50311', 'Waysdorf-tweets': 'WBS-43788' }



def import_Excel(inputFile):
    """Reads in an Excel-formatted spreadsheet as Pandas dataframes
    Links account records to a budget and vice versa via dicts
    returns: (dfBudget,dfAccount, dict account[budget], dict budget[account])
    NB: account[budget] values are lists as a budget might be used by more than one account
    """

    # hack: read_excel warns about an unsupported extension, suppress this warning
    warnings.filterwarnings('ignore', category=UserWarning, module='openpyxl')
    source = pd.read_excel(inputFile)
    # verify that spreadsheet left-most columns comply with known column list
    for col in range(len(COLUMNS)):
        if source.columns.array[col] != COLUMNS[col]:
            raise AssertionError('At column ' + str(col) + ': expected "' + COLUMNS[col] +
            '", got "' + source.columns.array[col] + '"')
    # classify rows
    #   budget:           column budget has a value, valid_from has a value
    #   account:          column account has a value, column budget has no value
    #   grand-total:      column unit is empty
    #   budget allocation: columns budget, account have no value, column valid_from has a value
    sourceRows = source.index.array
    dfBudget = source[ (source[C_BUDGET].notna() & source[C_VALIDFROM].notna()) ]
    budgetRows = dfBudget.index.array
    dfAccount = source[ (source[C_BUDGET].isna()  & source[C_ACCOUNT].notna()) ]
    accountRows = dfAccount.index.array
    grandtotalRows = source[ (source[C_UNIT].isna())].index.array
    if len(grandtotalRows) != 1:
        raise AssertionError('Expected one grand-total row, got ' + str(len(totalRows)))
    allocRows = source[ (source[C_BUDGET].isna() & source[C_ACCOUNT].isna() &
source[C_VALIDFROM].notna()) ].index.array
    # tag each row with type, check that our categorization is complete
    source[C_TYPE] = T_UNCLASSIFIED
    cat = {}
    for i in budgetRows:
        source.loc[i,C_TYPE] = T_BUDGET
    for i in accountRows:
        if source.loc[i,C_TYPE] != T_UNCLASSIFIED:
            raise AssertionError('Unable to categorize source data into budgets and accounts')
        source.loc[i,C_TYPE] = T_ACCOUNT
    for i in grandtotalRows:
        if source.loc[i,C_TYPE] != T_UNCLASSIFIED:
            raise AssertionError('Unable to categorize grand total in source data')
        source.loc[i,C_TYPE] = T_TOTAL
    for i in allocRows:
        if source.loc[i,C_TYPE] != T_UNCLASSIFIED:
```

```
            raise AssertionError('Unable to categorize budget allocations in source data')
        source.loc[i,C_TYPE] = T_ALLOC
    if len( source[(source[C_TYPE] == T_UNCLASSIFIED)] ) > 0:
        raise AssertionError('Could not categorize some input data rows: ' + str(otherRows))
    return source


def get_lookups(data):
    """ creates lookup dicts to find accounts with budgets and vice versa

    returns budget2account = dict with budget as key and list of accounts as value
            account2budget = dict with account as key and budget as value
    """

    budget2accounts = {}    # value is array because there can be >1 account with a budget
    account2budget = {}
    budgetRows = data[(data[C_TYPE] == T_BUDGET)]
    for i in budgetRows.index.array:
        df = data.iloc[i]
        budget = df[C_CODE]
        isNaN = df.isna()
        if not isNaN[C_ACCOUNT]:
            account= df[C_ACCOUNT]
        else:
            account = ''
        if budget in budget2accounts:
            if not account in budget2accounts[budget]:
                budget2accounts[budget].append(account)
        else:
            budget2accounts[budget] = [account]
        account2budget[account] = budget
    return (budget2accounts, account2budget)


def usage_per_account(data, account2budget):
    """ returns a dict with per account the used units
    """

    usage = {}
    for account in sorted(account2budget):
        budget = account2budget[account]
        dfusage = data[(data[C_TYPE] == T_BUDGET) & (data[C_CODE] == budget)]
        used = dfusage[C_USAGE].squeeze()
        usage[account] = used
    return usage


def show_usage_per_account(data, account2budget):
    """ creates a print-friendly table of use units per account, linked to wbs
```

```
    """

    out = 'COST CENTER(ACCOUNT) : USAGE IN CREDITS\n'
    usage = usage_per_account(data, account2budget)
    for account in sorted(usage):
        if account in ACCOUNT2WBS:
            wbs = ACCOUNT2WBS[account]
            out = out + wbs + '(' + account + ') : ' + str(round(usage[account])) + '\n'
    return out

def main(inputFile):
    try:
        data = import_Excel(inputFILE)
    except AssertionError as e:
        print('Input file format error: ' + str(e))
    (budget2accounts, account2budget) = get_lookups(data)

    print(show_usage_per_account(data, account2budget))

# main program
if __name__ == "__main__":
    main(INPUTFILE)
```