Salesforce Polling Service Scope Creep

Context

The current version of Salesforce Polling Service was written and maintained between April, 2017 and February, 2018 by only one employee who no longer works here.

With minimal documentation in the code, none on Confluence, and no one employed who is an expert on the code base, making changes is difficult because it is time-consuming to do and potentially dangerous to the operation since there are no rigorous tests.

There are other technical issues: the project's code is old and does not follow the application standard we use now, it is platform specific to Windows (.NET Framework), and is in need of refactoring due to two-way coupling between Salesforce and Everest OWC.

It is understood that this code has a limited lifespan: until full migration to D365. However, there are issues with the software (BES-2100) that should be addressed and important additions that need to be added (BES-2212).

My task was to go in and add order imports from the EU and CAN companies to Salesforce in an agile way, by extending with a quick and dirty approach and to make it work with acceptable levels of error. My efforts to do so thus far reveal a deep two-way dependency on specifically the EVEREST_OWC database.

As this ticket continues to expand in scope, I would like to discuss with the team on how to tackle this issue.

What I’m Currently Doing

I have been refactoring existing code, leaving most logic alone and adding connections to the EVEREST_OWCEU and EVEREST_OWCCAN databases.

There are two doctrines of refactoring that I’ve come up with:

1. For every SQL stored procedure that queries EVEREST_OWC, modify the command so that it also union selects EVEREST_OWCEU and EVEREST_OWCCAN.
    a. Some of these are already very large SQL queries and may result in unexpected behavior.
    b. It will require the modification of some Salesforce Polling Service specific tables.
    c. This could unexpectedly affect US orders propagating into Salesforce, which I don’t want to break even more.
    d. While this is likely the easiest solution, it is a blind one; it provides us even less diagnostics with already struggling software.
    e. There are points in the code that may require knowing which company a Salesforce object is from, meaning this solution may not even work.
    f. There could be document ID collisions.
2. The option I’m currently pursuing; going through the entire polling process step by step, wrapping US specific code with company-specific stored procedures (leaving US the same, and adding new stored procedures pointing to EU and CAN), adding additional logging, and analyzing points of failure, which also passively works on BES-2100.

a. I believe it is much safer, as it makes minimal changes to the US orders into Salesforce logic.
b. I can possibly solve BES-2100 in the process of working through it.
c. I will gain a very good understanding of the connection between Everest and Salesforce which will become very useful when connecting D365 and Salesforce later.
d. Combing through all of Salesforce Polling Service, I will become experienced with the codebase and can continue to work on the project; presumably more issues will arise and more improvements will be wanted.
e. This is a much longer process than option 1, but one that I think is worthwhile.

Recommendation

While there may be more refactoring options that I have not thought of, and am open to considering. With my current understanding of Salesforce Polling Service, I think the best solution for both satisfying the requirements of the software and minimizing time is to write it new, from scratch.

If writing from scratch using paradigms we currently follow as a team, this will allow anyone to become accustomed with the code base. If assigned to this project I will ensure that the code is clear and well documented.

I recommend we spend time analyzing the current flow of information between Everest and Salesforce. We should also take input from Andrew on how this could be better. Using this information, design an improved flow of information. We should also research the query loads, and create intentional (and possibly dynamic) polling intervals instead of something seemingly arbitrary like the current batch of 200 records every 3 minutes.

I also recommend that we use a adapter design pattern for the ERP side of the flow so that when the full migration to D365 comes around, this software does not become useless and could possibly even be supplemental while we're still looking for or installing an existing or proprietary connector for D365 and Salesforce.

Potential User Stories

- As a sysadmin, I want to be able to control the Salesforce Polling Service for a specific company/region so maintenance can occur or adjustments to the software can be made.

    - Turn it on or off if there needs to be database maintenance for a specific company/region
- As a developer, I want to be able to add support for different ERP systems so I don't have to make new software.
    - Abstract away ERP representations with an intermediate facade
- As a sysadmin, I want to be able add ERP import support for more companies/regions in the future.
    - Containerized
- As a sales rep, I want to know which company/region the customer/account/order is from so I can provide better support.
- As a sysadmin, I want the Salesforce Polling Service to scale with load changes dynamically so that in low load times it isn't wasting resources and in high load times it can account for the volume.
- As a database admin, I don't want the Salesforce Polling Service to put unnecessary load on the database.
- As a sales rep/customer service agent, I want customer/account/order information to consistently and automatically be present in Salesforce so I don't have to lose time manually checking the database.
- As a sysadmin, I would like to know when and where an error occurred in the code and what it means so I can quickly diagnose the problem.
    - This could go as far as numbering and documenting error conditions as they occur into Confluence
- As a developer, I want there to be up to date documentation on the code so that it is easy to service.