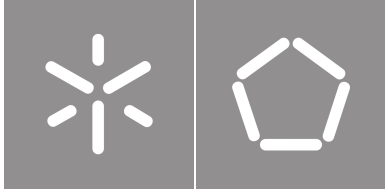
Universidade do Minho
Escola de Engenharia

André Tiago Gonçalves Ramalho

Automação de Testes

Junho de 2022



Universidade do Minho
Escola de Engenharia

André Tiago Gonçalves Ramalho

Automação de Testes

Dissertação de Mestrado
Mestrado Integrado em Engenharia Informática

Trabalho efetuado sob a orientação de:
Rui Manuel Ribeiro Castro Mendes

Junho de 2022

DIREITOS DE AUTOR E CONDIÇÕES DE UTILIZAÇÃO DO TRABALHO POR TERCEIROS

Este documento foi gerado com o processador (pdf/Xe/Lua)LaTeX e o modelo NOVAthesis (v6.9.15) [7].

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração do presente trabalho académico e confirmo que não recorri à prática de plágio nem a qualquer forma de utilização indevida ou falsificação de informações ou resultados em nenhuma das etapas conducente à sua elaboração.

Mais declaro que conheço e que respeitei o Código de Conduta Ética da Universidade do Minho.

Braga, 30 de Junho, 2022
(Local) (Data)

André Tiago Gonçalves Ramalho
(André Tiago Gonçalves Ramalho)

Resumo

Automação de Testes

De modo a minimizar o tempo expendido nos testes das várias funcionalidades do *software* Systems, Applications and Products in Data Processing (SAP), bem como aumentar o número de testes e combinações de dados de entrada possíveis, foi proposto pela Accenture o desenvolvimento de um BOT que execute automaticamente os mesmos.

Para tal, comecei por familiarizar-me com o SAP Human Capital Management (HCM), um programa responsável pela gestão de recursos humanos, bem como aprender os passos para realizar os testes manualmente.

A seguir, optei por ler a documentação do AutoHotKey (AHK) e Selenium visando aprender a usá-las.

AHK é uma linguagem *scripting* para Windows utilizada para automatizar tarefas. Foi usada para interagir com o programa SAP HCM. Já o Selenium WebDriver também foi usado para automação, mas relacionado com a automação do *browser*, tendo sido utilizado para automatizar um Portal WEB.

Fiz 25 testes, em que executam várias tarefas e comparam resultados entre os testes "antes"e "depois". A título exemplificativo podemos executar como teste a criação de um registo de um determinado conjunto de dados e também executar um teste que permita obter a lista de registos e comparar se os resultados obtidos estão corretos.

Por último, dediquei-me ao desenvolvimento do BOT com o objetivo de criar uma solução capaz de realizar os testes automaticamente e que seja possível adicionar e alterar os testes rapidamente de modo a aumentar a produtividade na Accenture.

Palavras-chave: Automação de Testes, AutoHotkey, Selenium, SAP Software Solutions

Abstract

Tests Automation

In order to minimize the time spent on testing the various functionalities of the SAP software, as well as increase the number of possible tests and combinations of input data, Accenture proposed the development of a BOT that performs these tests.

To do this, I started by familiarizing myself with SAP HCM, a program responsible for human resource management, as well as learning the steps to perform the tests manually.

Next, I chose to read the AHK and Selenium documentation in order to learn how to use them.

AHK is a *scripting* language for Windows used to automate tasks. It was used to interact with the SAP HCM program. Selenium WebDriver was also used for automation, but related to the automation of the *browser*, and was used to automate a WEB Portal.

I did 25 tests, in which they perform various tasks and compare results to check if the output changed between tests. As an example we can run as a test the creation of a record for a given data set, and also run a test to get the list of records and compare if the results obtained are corrected

Finally, I dedicated myself to the development of the BOT with the objective of creating a solution capable of performing the tests automatically and be able to add and change tests quickly in order to improve the productivity at Accenture.

Keywords: Test Automation, AutoHotkey, Selenium, SAP Software Solutions

Índice

Índice de Figuras

Índice de Tabelas

Índice de Listagens

Glossário

BOT BOT é um software programado para automatizar tarefas repetitiva, frequentemente imitando ou substituindo o comporamento de seres humanos. [4] *(pp. iv, v, 1, 12, 16–18, 25, 26, 29, 33)*

bug É um erro ou uma falha numa aplicação que faz com que a aplicação produza resultados inesperados *(p. 3)*

Histórias de Utilizador Uma história de utilizador é uma descrição informal, de uma ou mais características de um sistema de software na perspectiva de utilizador final. Uma história deve responder qual é o tipo de utlizador, o que pretende e porquê [10]. Tem a seguinte estrutura: como um <tipo de utlizador >, eu quero <algum objetivo > de modo que <alguma razão >. *(p. 3)*

rubrica código que define uma determinada parcela no recibo de vencimento *(pp. 37, 38)*

unidade organizacional código numérico que correspodne a um departamento de uma empresa dentro do SAP ERP *(pp. 36, 37)*

Siglas

AHK AutoHotKey *(pp. iv, v, 13, 14)*

COM Component Object Model *(pp. 14, 16)*

HCM Human Capital Management *(pp. iv, v, 1, 12, 16, 23)*

SAP Systems, Applications and Products in Data Processing *(pp. iv, v, 1, 12, 14, 16, 23)*

1 Introdução

1.1 Enquadramento

A verificação da qualidade do código escrito sempre foi algo crítico e importante no desenvolvimento de *software*. No entanto, é uma atividade que, não só necessita demasiado tempo dispensado[3], mas também está sujeito ao erro humano[5].

Deste modo, a substituição das tarefas por automatismos é uma aposta interessante, uma vez que resolve os problemas referidos acima e permite melhorar a qualidade[5] e a quantidade de testes efetuados.

1.2 Objetivos

Este projeto tem como objetivo o desenvolvimento de uma nova ferramenta (BOT) capaz de realizar testes aplicacionais de forma autónoma, considerando integração com diferentes sistemas, diferentes módulos SAP HCM e condições de teste variáveis. Além da realização dos testes, a ferramenta deverá ter também a capacidade de guardar resultados (*output* dos testes), comparar testes (antes e depois), confirmar resultados e avaliar a qualidade dos mesmos (sucesso e insucesso).

Para além de atividades técnicas associadas à solução tecnológica, será objetivo deste projeto integrar a vertente do negócio e mercado, dando uma perspetiva global entre a realidade tecnológica e a realidade de negócio e experiência de utilização.

2

Estado da Arte

2.1 Introdução

Os Seres Humanos no seu dia a dia e na sua atividade profissional, estão sempre sujeitos a cometerem erros, que podem ser causados por diversos fatores, como, por exemplo, distração ou pelo facto de os requisitos não serem claros o suficiente. Os erros são ainda possíveis de acontecer devido à falta de tempo, de conhecimentos técnicos, ou por suposições erradas. Estes erros, por vezes, podem não ser preocupantes, contudo há situações em que podem acarretar graves consequências, nomeadamente perda monetária, perda de confiança do *software* por parte dos utilizadores e/ou investidores, ou até implicar o custo de vidas humanas. Por esta razão, testar correta e rigorosamente é uma etapa muito importante no desenvolvimento de *software*.

Neste sentido, executar todos os testes necessários no *software* é de extrema importância, na medida em que permite encontrar anomalias e falhas no *software*, através de *inputs* com dados artificiais, antes que seja utilizado pelos clientes.

Além disso, possibilita ainda mostrar com maior confiança o que este consegue fazer e a sua fiabilidade, apesar de não serem 100% fiáveis, pois, tal como Dijkstra disse numa conferência patrocinada pela NATO em 1969: "testes apenas mostram a presença de erros e não a sua ausência [6]".

Neste sentido, quanto mais completos forem os testes, menor a probabilidade de ocorrer uma falha ou comportamentos imprevistos na execução do código compilado.

De forma a provar que o programa é funcional, devem ser, pelo menos, testados todos os requisitos funcionais e não funcionais do *software*, ou seja, todas as funcionalidades implementadas no *software*[13].

2.2 Tipos de teste

2.2.1 *Static Testing*

Ora, para o *software* ser testado com o rigor exigido, têm sido desenvolvidas várias formas de testar, como é o caso de *static testing*. Este é um tipo de teste onde o *software* é testado sem ser executado nem compilado, sendo realizado durante o estágio inicial de desenvolvimento.

O *software* é examinado através da análise e depuração do código-fonte e da sua arquitetura e através da análise dos requisitos funcionais e Histórias de Utilizador.

Este tipo de testes permite que os problemas sejam detetados mais cedo e potencialmente reduzir o custo e o tempo da produção do *software*. Além disso, também pode detetar potenciais problemas que não possam ser detetados durante *dynamic testing* (referido em 2.2.2), como, por exemplo, a existência de variáveis não inicializadas ou não utilizadas.

Para além de bugs, também pode encontrar falhas de segurança no código, ou encontrar possíveis melhorias no desempenho do *software* através do aprimoramento do *design*, como, por exemplo, encontrar problemas na eficiência dos algoritmos ou na estruturas de bases de dados.

Outra distinção entre *static* e *dynamic testing*, é o primeiro ser usado para encontrar defeitos diretamente no código, em vez de encontrar falhas causadas pelos defeitos do *software* enquanto está em execução. Um defeito pode existir durante muito tempo sem causar nenhuma falha.

Visto que o código do SAP não é desenvolvido pela Accenture e não se tem acesso ao mesmo nem aos requisitos, *static testing* não irá ser usado nesta dissertação.

Existem 2 técnicas para colocar *static testing* em prática: *review* e *static analysis*.

2.2.1.1 Review

Static testing review permite encontrar falhas em documentos, como, por exemplo, do desenho do *software* e da lista de requisitos. Está classificado em 4 partes:

- **Informal:**
  Tal como o nome indica, este tipo de revisão é informal e não segue nenhum processo para encontrar problemas. Geralmente membros da equipa reúnem-se e dão comentários acerca do produto.

- **Walkthrough:**
  O autor explica o produto e outros membros da equipa expõem as suas dúvidas, caso existam. Este tipo de *review* são informais.

- **Peer review:**
  Os documentos são verificados pelos colegas do autor, a fim de avaliar a qualidade do mesmo.

- **Inspection:**
  Uma reunião é conduzida por um moderador treinado. Neste tipo de revisão, é seguido um processo rigoroso. Os revisores seguem uma lista para encontrar os defeitos.

2.2.1.2 Static Analysis

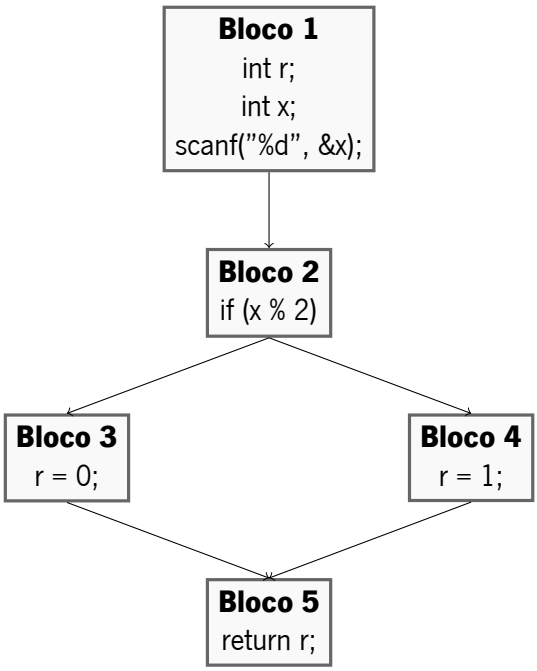
Durante *static analysis*, é avaliado o código e é detetado, por exemplo, código e variáveis que não estejam a ser usadas, variáveis não definidas, vulnerabilidades de segurança e *Coding Standard Violations* e *loops* infinitos.

Existem 3 tipos de *Static Analysis*:

- ***Control Flow***
  É uma técnica de testes usada para determinar a ordem de execução de declarações ou instruções do programa através de uma estrutura de controlo.

  Control Flow Graph (G) é definido por um conjunto de *nodes* (N) e *edges* (E), onde um *edge* E(i,j) conecta os nodes Ni e Nj

  Por exemplo:




```
N = {1, 2, 3, 4, 5}
E = { (1,2), (2,3), (2,4), (3,5), (4,5) }
```

- ***Data Flow***
  Usa o *Control Flow* para analisar o fluxo do código em relação às variáveis. As varíaveis podem estar:

- Definidas, mas não usadas
- Usadas, mas nunca definidas
- Definida mais que uma vez

Usando o mesmo exemplo do *Control Flow*:

Variável	Bloco Definido	Usada
x	Bloco 1	Bloco 2
r	Bloco 1	Bloco 3, 4 e 5

- ***Cyclomatic Complexity***

  Métrica usada para indicar a complexidade de um programa utilizando o *Control Flow*

  Cyclomatic Complexity é dado pela letra m e é defina como:

  ```
  M = E - N + 2P
  ```

  onde,

  ```
  E = número de edges in the control flow graph
  N = número de nodes in the control flow graph
  P = número de componentes conectados
  ```

2.2.2 *Dynamic Testing*

Outro meio desenvolvido para ajudar na correta testagem do *software*, foi o *Dynamic Testing*, em que durante os testes o *software* é verificado através da execução do código já compilado. É testado com determinados *inputs* e o *output* é analisado. No decorrer da realização deste tipo de testes é possível constatar se o *software* funciona corretamente e também o desempenho e recursos de *Hardware* necessários. Este é o tipo de testes que será usado nesta dissertação.

Ao contrário do *Static Testing*, o *Dynamic Testing* é utilizado numa fase mais tardia do desenvolvimento do *software*.

Assim, o *Dynamic Testing* pode ser classificado em 2 categorias:

- White-box: todo o código é visível durante o teste. É testada a estrutura do código, como ciclos, condições e estrutura dos dados e do código. Geralmente é testado pelos programadores do *software* ou por alguém com conhecimentos de programação e conhecimentos sobre a estrutura e *design* do *software* a ser testado.

- Black-box: é assumido que o *software* é uma caixa negra onde apenas se sabe quais são os *input* e *outputs*. Apenas são testadas as funcionalidades e requisitos funcionais do *software* e o seu comportamento. O que o programa faz é internamente ignorado e é testado na perspetiva do utilizador.

  Existem várias técnicas para *Black-box testing*, como por exemplo:

  - *Equivalence Partitioning*

    Divide um conjunto de *inputs* em classes que tenham o mesmo comportamento com o objetivo de reduzir o número de testes.

    De seguida testa pelo menos 1 membro de cada classe e assume que o comportamento irá ser o mesmo para os restantes *inputs* da mesma classe.

    Vamos usar como exemplo uma aplicação de um estabelecimento que contenha menus diferentes dependendo das idades dos clientes.:

    * Menu Infantil dos 6 aos 11 anos
    * Menu Jovem dos 12 aos 17 anos
    * Menu Adulto dos 18 aos 64 anos
    * Menu Sénior para mais de 65 anos

    Nesse caso, os inputs seriam divididos pelas seguintes classes:

	Inválido	**Infantil**	**Jovem**	**Adulto**	**Sénior**
Idade	<6	6-11	12-17	18-64	>65
Classe	Classe 1	Classe 2	Classe 3	Classe 4	Classe 5

Tabela 1: Exemplo de *Equivalence Partitioning*

  - *Boundary Testing*

    Testa os limites entre classes de *input* uma vez que é um erro bastante comum na programação (por exemplo, usar > em vez de >=)

    Usando o exemplo anterior, seria testado para as idades 5, 6, 11, 12, 17, 18, 64 e 65 anos.

  - *Decision Table Testing*

    É uma tabela que permite relacionar as várias combinações de *input* e os valores lógicos (verdadeiro ou falso) e a ação correspondente.

    Por exemplo, para testar o *login* num website:

Condições	Regra 1	Regra 2	Regra 3	Regra 4
Username	F	V	F	V
Password	F	F	V	V
Ação	Erro	Erro	Erro	*Login*

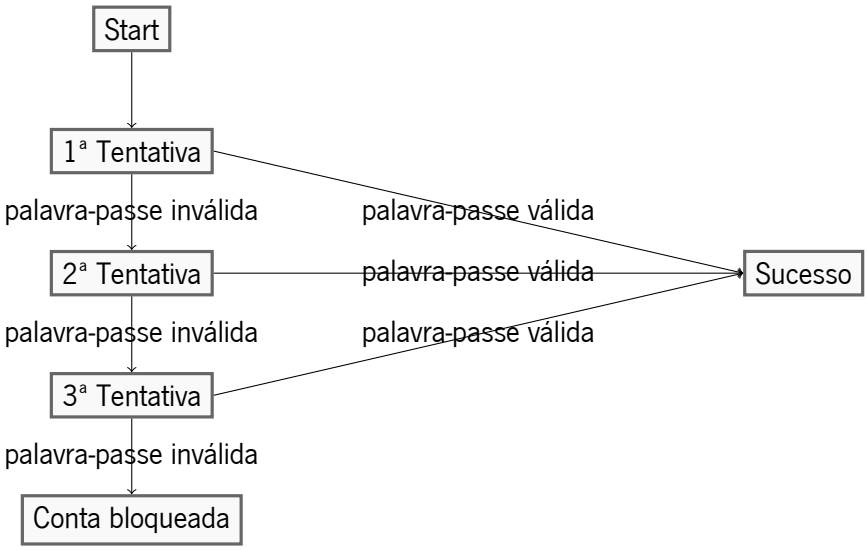
Tabela 2: Exemplo de *Decision Table Testing*

- *State Transition Testing*

  Em alguns sistemas, são geradas respostas significativamente diferentes quando o sistema transita de um estado para outro.

  State Transition Testing é um tipo de teste em que alterações nas condições de *input*, implicam alterações do estado ou do *output*. Neste tipo de testes, o comportamento do sistema é observado, tanto os valores de entrada positivos como negativos.

  Voltando ao exemplo do início de sessão de um utilizador, vamos agora considerar que se o utilizador errar a palavra-passe mais de 3 vezes, o sistema transita para um estado diferente, bloqueando a conta.



- *Error Guessing*

  Permite identificar erros baseados em experiências em testes anteriores e na intuição.

  Falhas comuns são, por exemplo:

  * Lixo no input
  * Input vazio
  * Apontadores vazios
  * Input demasiado grande
  * Dividir por 0

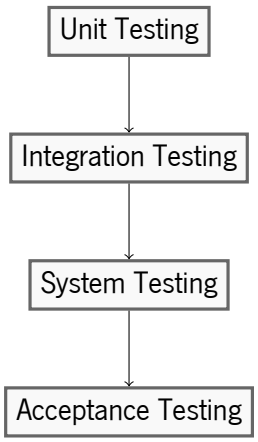
Dentro da categoria de *Black-box testing*, existem testes funcionais e não funcionais.

2.2.2.1 Testes Funcionais

Os testes funcionais são usados para testarem se as funcionalidades desenvolvidas estão de acordo com as especificações funcionais definidas.

Os testes funcionais podem ser divididos da seguinte forma:

- Unit Testing: é feito pelos programadores para verificarem que parte do código escrito está a funcionar. Estes testes são feitos durante o desenvolvimento da aplicação
- Integration Testing: frequentemente aplicações contém múltiplos módulos desenvolvidos por diferentes programadores. O objetivo deste teste é verificar que estes interagem e integram corretamente.
- System Testing: é testado o software inteiro e tem o objetivo de verificar se cumpre todos os requisitos funcionais e não funcionais.
- Acceptance Testing: é verificado se o *software* cumpre os requisitos comerciais para ser utilizado. Não é testado para encontrar falhas ou bugs nas funcionalidades, mas para testar características não funcionais como a usabilidade. Geralmente o teste é feito pelos utilizadores finais.



2.2.2.2 Testes não Funcionais

Alguns exemplos de testes não funcionais são:

- Desempenho;
- Compatibilidade;
- Segurança;
- Facilidade de utilização.

2.3 Automação de Testes

A automação de testes procura substituir a atividade humana e usar ferramentas e *scripts* para substituir atividades repetitivas, com o objetivo de melhorar a eficiência dos testes.

Com o uso de *software* é possível automatizar a execução de testes, comparando os resultados esperados com os resultados reais.

Existem várias razões para optar pela automação dos testes[8].

- Possibilidade de executar os testes com maior frequência;
- Efetuar testes de regressão para garantir que não existem novos defeitos em relação à versão anterior;
- Efetuar testes que não são possíveis ou viáveis de fazer manualmente;
- Efetuar sempre os mesmos testes é uma tarefa muito repetitiva e monótona. Libertar os trabalhadores a realizar desses encargos pode melhorar o contentamento dos trabalhadores;
- Os *scripts* podem ser reutilizáveis.

Por outro lado, a automação também tem alguns problemas e limitações, tais como:[8]

- Má prática de testes. Se os testes não estiverem bem organizados e definidos ou se tiver pouca documentação ou inconsistente, e os testes não forem bons a encontrarem defeitos, os testes automáticos também não terão muito sucesso;
- Manutenção dos Testes. Quando o *software* é alterado, é muito frequentemente necessário também alterar os testes. A automação é útil se o esforço necessário para atualizar os testes for menor que correr os testes manualmente;
- Não substitui os testes manuais. Haverá sempre alguns testes que devem continuar a ser feitos manualmente, como, por exemplo:
  - Testes que são raramente executados;
  - Quando o *software* é muito volátil, poderá não ser viável automatizar um teste se é raramente executado ou se o *software* é muito volátil, ou seja, se a interface ou a funcionalidade for alterada muito frequentemente, provavelmente não será eficaz em termos de custos;
  - Testes que sejam facilmente verificados por humanos, mas difíceis ou impossíveis de automatizar;
  - Testes que implicam interações físicas.

- Ferramentas não têm imaginação e apenas seguem um conjunto de passos definidos pelo programador. Se, por exemplo, ocorrer um comportamento inesperado ao executar um teste, a ferramenta pode não ter um comportamento definido para lidar com essa situação, e retornar um erro, enquanto o ser humano pode usar a sua criatividade para contornar a situação inesperada.

A automação de testes pode ser executado em 3 diferentes níveis:

- *Unit Level Automation*
  Tal como já foi referido em 2.2.2, é um tipo de testes onde componentes são individualmente testados. *Unit tests* isolam uma secção do código e procuram identificar os problemas;

- *API Testing*
  Application Programming Interface (API) é uma interface que permite a comunicação e troca de dados entre 2 softwares.

  Este tipo de testes verifica a funcionalidade, confiabilidade, desempenho e segurança de APIs;

- *User Interface*
  Usado para identificar defeitos no produto utilizando a *interface* gráfica de utilizador (GUI).

Quanto maior a complexidade do software e a quantidade de funcionalidades, maior tempo é necessário para testar todas as funcionalidades, sendo, por vezes, boa ideia investir na automatização.

2.3.1 Automação de Testes da Interface Gráfica

Softwares que utilizam interface gráfica e não são invocados pelo terminal são mais difíceis e complexos de automatizar, uma vez que é preciso lidar, por exemplo, com a diferentes resoluções do ecrã, cliques do rato, combinações de teclas, arrastar o rato para efetuar ações[15].

De forma a criar *scripts* eficientes e robustos para testar a interface gráfica, é recomendado evitar usar as coordenadas do ecrã, pois estas podem diferir conforme, por exemplo, a resolução, plataforma e tema. Em vez disso, deve-se usar valores únicos para localizar *GUI Components*.

Muitas vezes programadores usam o mesmo nome para vários GUI Componentes ou então são gerados automaticamente pelo IDE[15], no entanto valores *text labels*, *button captions*, *window titles*[15] são geralmente únicos. Também é possível usar uma combinação das várias propriedades dos componentes.

2.3.2 Keyword driven

Keyword driven é uma forma de automação que usa keywords para descrever um conjunto de ações, ou seja, cada keyword é interpretado e associado a uma ação que executa um conjunto de passos. Este método possibilita a construção de testes independentes da *Framework* numa linguagem acessível a qualquer pessoa.

Por exemplo, para descrever o processo de efetuar uma encomenda poderíamos ter as seguintes keywords:

Keywords	Ação
Login	Efectua o login
AdiconarProduto	Adiciona produtos ao carrinho
FinalizarEncomendas	Fazer checkout

Algumas vantagens deste método são:

- Possibilidade de reutilizar keywords para novos testes.
- Os testes são fáceis de ler e de modificar.
- Apenas a equipa que implementa keywords precisa de ter conhecimentos de programação.
- A construção de cenários de testes são fáceis de criar por não estarem dependentes de uma linguagem de programação.

2.3.3 Data driven

Data driven é um método de teste em que é testado o mesmo teste várias vezes, mas com diferentes dados de input. De forma a simplificar o processo, os dados input são guardados num ficheiro auxiliar, como tabelas, *spreadsheet* ou base de dados. Isto permite testar cenários com múltiplos dados de input e vários cenários. A título de exemplo, no caso da autenticação de um utilizador, é importante testar ambos os cenários em que o utilizador e a palavra-passe estão corretos e errados.

Também pode ser útil para testar várias vezes o mesmo teste com vários inputs para encontrar regressões durante a fase de desenvolvimento.

2.4 Vertente de Negócio

Antes de se optar pela automação, é geralmente boa ideia estar consciente das vantagens, desvantagens e limitações da automação referidas na secção 2.3, pois desta forma é que se poderá saber se optar pela automação será ou não viável.

A razão que a Accenture optou pela automação das funcionalidades do SAP foi para reduzir o tempo necessário para executar cada teste, o que permite também aumentar o número de inputs usados para cada teste, e o número de testes.

No entanto, também é importante não esquecer que, apesar de o tempo para executar seja menor, ainda existe o tempo despendido para implementar a automação de cada teste.

Por essa razão, nesta dissertação, optou-se por se focar no tempo necessário para implementar a automação de cada teste. Por exemplo, caso exista algo trabalhoso ou não imediato de automatizar,

mas se um ser humano conseguir verificar imediatamente comparando as capturas de telas no relatório gerado pelo *BOT*, então não fará sentido automatizar o resultado do teste e apenas indica que o resultado é desconhecido. Além disso, também reduz as habilidades necessárias do programador para fazer a manutenção futura dos testes do BOT caso estes deixem de funcionar devido a alterações significativas do Systems, Applications and Products in Data Processing (SAP) Human Capital Management (HCM).

3

Implementação

3.1 Introdução

O objetivo do BOT desenvolvido é automatizar os testes de várias transações feitas no programa SAP HCM e também vários testes num Portal WEB. Neste caso, coloquei em prática a execução de 23 testes, 2 dos quais realizados para o programa SAP HCM e 2 testes para o Portal WEB (Figura 4).

Os testes são feitos a nível da GUI, ou seja, o BOT terá de ser capaz de enviar comandos e interagir com a Interface Gráfica do Programa SAP e do Portal WEB.

Todos os testes efetuados usam a técnica *Black Box Testing*.

3.2 Ferramentas e Linguagens Utilizadas

3.2.1 AutoHotkey

AutoHotKey (AHK) é uma linguagem de *scripting* para Windows que permite automatizar tarefas.

Esta é a linguagem optada para o desenvolvimento do BOT uma vez que é simples de entender e usar, sendo utilizada para lançar e interagir com programas, como, por exemplo, clicar em botões, arrastar elementos com o rato, enviar texto para uma caixa de texto e enviar entradas do teclado, combinações de teclas.

AutoHotkey é *opensource* e usa a licença GNU GLv2[1], garantindo a liberdade de indivíduos, organizações e empresas de ler e modificar o código.

AHK é um *fork* de outra ferramenta de automação chamada AutoIT. Apesar de terem uma sintaxe diferente, ambas ferramentas são gratuitas a desempenham a mesma funcionalidade. No entanto, ao contrário do AutoHotkey, AutoIT tornou-se proprietário e *closed-source* em 2005.

De acordo com o Google Trends, AutoIT é, neste momento, menos popular que o seu *fork* AutoHotkey[14] (figura 1).

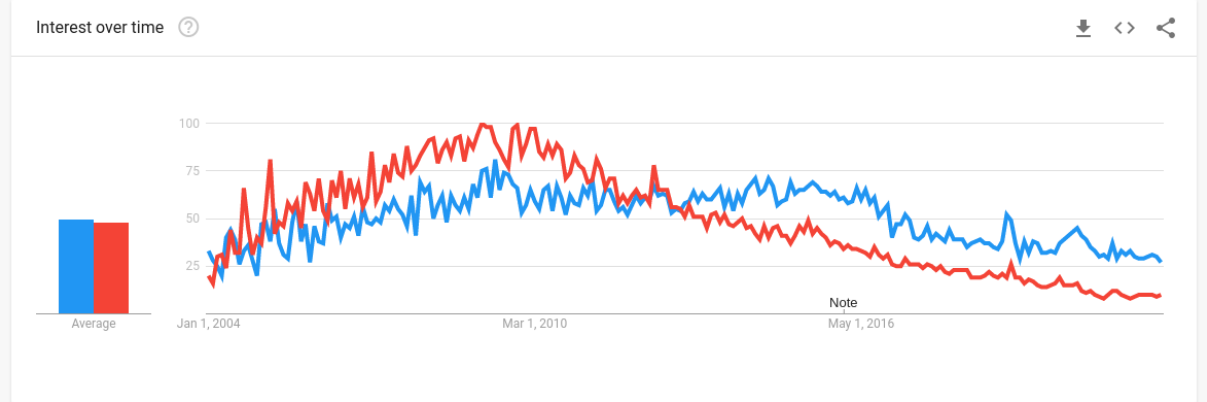


Figura 1: Interesse do AutoHotkey e AutoIT ao longo do tempo de acordo com as pesquisas efectuadas no google search

Pelas razões ditas em 2.3.1, deve-se evitar usar as coordenadas do ecrã. No entanto, se caso seja mesmo necessário, é recomendado utilizar o CoordMode" "Client" visto que é o que está menos dependente da versão do sistema operativo e tema utilizado[2]. Este modo usa as coordenadas relativas à janela ativa, o que exclui, por exemplo barra de título da janela e de menu, caso exista[2].

Infelizmente, não é possível ao AHK aceder diretamente a elementos individuais do SAP, ou seja, não é possível, por exemplo, clicar no botão com um determinado texto ou ID. No entanto, continua a ser possível ler as barras de título (útil para saber quando o estado da janela mudou), clicar em determinadas coordenadas do ecrã, enviar entradas do teclado e combinações de teclas, como, por exemplo, alterar o elemento em foco usando a tecla TAB.

Todavia, apesar de não conseguir interagir diretamente com o SAP, é possível concretizá-lo usando o SAP Gui Scripting API com o AHK através da interface Component Object Model (COM).

3.2.2 SAP Gui Scripting API

SAP Gui foi criado pela equipa do SAP para facilitar o controlo da interface do SAP e automatizar tarefas repetitivas.

O SAP permite gravar todas as interações efetuadas no programa e gerar um *script* escrito em Microsoft's VBScript que utiliza a API fornecida pelo SAP para aceder à interface gráfica do SAP.

Foi criado com os seguintes *use cases*[11]:

- Teste automático da funcionalidade SAP;
- Aplicações front-end personalizadas para substituir o *GUI* do SAP;
- Ferramentas para personalizar aplicações no nível SAP GUI -> GuiXT;
- Aplicações de E-Learning que guiam um utilizador através de transacções SAP.

3.2.3 Selenium

Selenium WebDriver é uma ferramenta open-source que permite automatizar *web applications*. Utiliza o *engine* de um *Browser*, como por exemplo Mozilla Firefox (gecko) ou Google Chrome (blink) para manipular diretamente as páginas web.

Além disso, também suporta múltiplos sistemas operativos como Windows, Linux e MacOS.

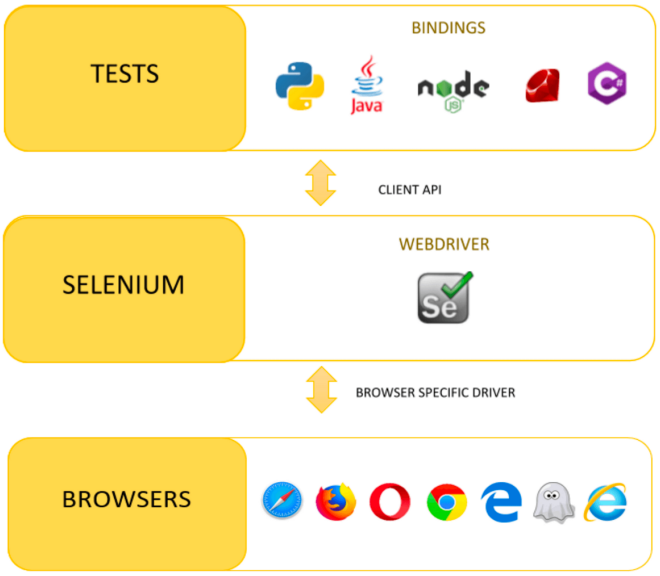


Figura 2: Arquitetura do Selenium. Imagem obtida no link https://www.blazemeter.com/blog/selenium-webdriver-for-beginners

Selenium WebDriver pode ser utilizado em muitas linguagens de programação como C#, Ruby, Java, Python e JavaScript.

Para utilizar o Selenium WebDriver é necessário[12]:

- Instalar a biblioteca do Selenium para a linguagem de programação pretendida
- Transferir o *driver* para o *browser* em que irá ser executado os testes. Este *driver* é o responsável por invocar funções nativas do *browser* para emular as operações do utilizador.
  A manutenção dos *drivers* é feita pelos próprios fabricantes dos respetivos *browsers*, sendo apenas o Internet Explorer a exceção, dado que a manutenção do mesmo é feita pelo Selenium Project.

Existe sempre alguma incerteza no futuro de qualquer *software*. Ao longo dos anos, vários *browsers* foram descontinuados, como o Netscape, Safari (apenas versão para Windows) e, muito brevemente, o Internet Explorer também irá deixar ser suportado[9]. Outros navegadores sofreram grandes alterações, como o caso do Opera e Microsoft Edge terem abandonado o seu *engine*, Presto e EdgeHTML respetivamente, a favor do motor da google, *Blink*.

Blink, por sua vez, é um *fork* de um componente do *WebKit*, *engine* desenvolvido e suportado pela Apple, que por sua vez é um *fork* do KHTML desenvolvido pelo KDE.

Por esse motivo, uma das grandes vantagens do Selenium WebDriver é de não estar dependente de apenas um navegador ou Sistema Operativo e não ser necessário reescrever código caso se pretenda, por exemplo, efetuar os testes noutro *Browser*.

3.3 Arquitetura

O BOT foi primariamente desenvolvido em AutoHotkey e contém uma simples interface em que o utilizador tem a possibilidade de selecionar os testes a serem executados bem como o login e password a ser usados nos testes (figura 4).

Toda a parte dos testes relacionados com a aplicação SAP HCM foi feito usando a linguagem *scripting* AutoHotkey que integra também o SAP GUI Scripting utilizando objetos COM.

Em relação ao Portal WEB, não foi usado o AutoHotkey porque a única forma de manipular com as páginas WEB seria usando COMs, que apenas funciona com o Internet Explorer ou enviando *Keystrokes* para interagir com as páginas.

Nenhuma dessas alternativas são ideais, uma vez que o Internet Explorer vai deixar de ser suportado pela Microsoft em 2022 [9] e se usar comandos do teclado para interagir com as páginas o BOT torna-se mais vulnerável a atualizações no UI do Portal WEB.

Por esse motivo, foi optado usar o Python com o Selenium para fazer a parte dos testes em que é usado um Portal WEB.

Com o Selenium, já é possível encontrar partes do código-fonte da página e interagir e obter a informação pretendida de forma mais precisa e eficaz.

A metodologia usado neste BOT é *data driven*, ou seja, para cada teste existe associado um ficheiro CSV que contém todos os valores a serem usados para cada teste.

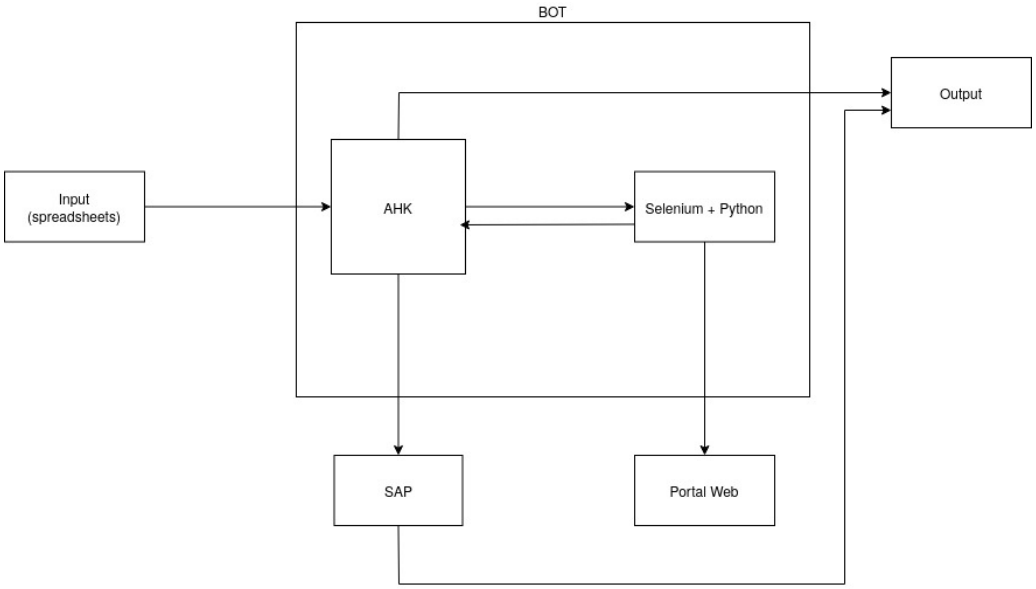


Figura 3: Funcionamento do BOT

3.4 Funcionamento

Quando o BOT é executado, apresenta ao utilizador uma janela para inserir o username e password do servidor SAP, e várias checkboxes para selecionar os testes a serem executados. Adicionalmente, também tem um botão para selecionar a conexão a ser usada, permitindo escolher qual o servidor SAP.

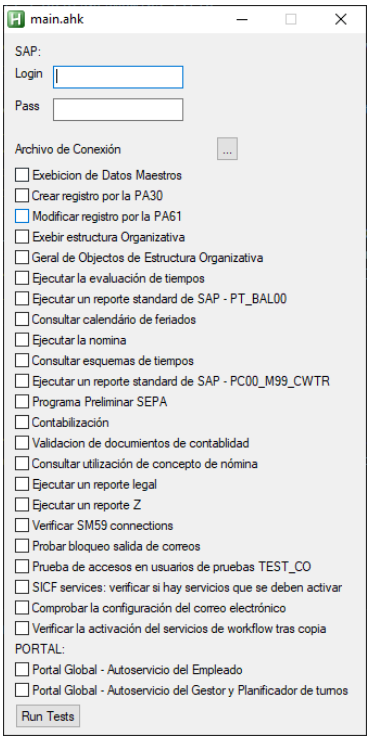


Figura 4: Interface do BOT

O código do BOT, tem, para cada teste, as seguintes variáveis internas associadas:

- **enabled**: indica se o teste está ou não selecionado e se será executado
- **title**: texto que aparece no GUI durante a seleção dos testes a serem executados.
- **type**:
  - SAP: caso o teste for executado no SAP
  - PORTAL: caso o teste seja executado no Portal Web
- **hasData**: *true* caso exista uma planilha de dados com os parâmetros a serem usados no teste.

O projeto do BOT tem a seguinte estrutura:

```
BOT
├── connections
├── data
├── output
└── tests
    ├── sap
    └── portal
```

- A diretoria **connections** contém ficheiros `.sap` que são atalhos que o BOT usa para conectar ao servidor que se pretende ser testado, ou seja, para cada servidor existe um ficheiro `.sap`.
- A diretoria com o nome **data** que contém `spreadsheets` com os valores que serão utilizados no teste.

  Por exemplo, se quiser executar uma transação para adicionar os empregados com o número 123456 e 456789 com licença de maternidade do dia 01/08/2020 a 01/12/2020 e 05/05/2021 a 05/09/2021 respetivamente, usaria o seguinte ficheiro CSV:

	A	B	C	D	E
1	Empleado	Infotipo	Subtipo	Desde	Até
2	123456	2001	0130	01.08.2020	01.12.2020
3	456789	2001	0130	05.05.2021	05.09.2021

Figura 5: Exemplo de dados de uma *Spreadsheet* usada para um dos testes

- A pasta **output** é a localização de todos os ficheiros criados pelo BOT. Isto inclui PrintScreens, ficheiros de texto exportados do SAP e as diferenças com o teste "antes"caso existam e o relatório report.html gerado. Este relatório inclui o resultado obtido para cada teste e os *screenshots* associados a cada teste.

- Na diretoria **tests/sap** e **tests/portal** existem os scripts a serem executados em cada teste do SAP e no Portal respetivamente. Também existe o ficheiro `tests/sap/functions.ahk` que contém funções que são comuns a vários testes do SAP.

O BOT usa o AutoHotkey para lançar o SAP HCM e interagir com o mesmo e depois lê os resultados obtidos utilizando a funcionalidade *export* do SAP HCM, através do SAP GUI Scripting API ou do *clipboard* do sistema operativo (ctrl+C) e verifica se os resultados obtidos estão corretos e se as operações foram executadas com sucesso.

Já no caso do Portal WEB, a interação e a leitura dos resultados é feito com o Selenium, através da localização de elementos da página web.

Quando o BOT executa um teste, é possível ocorrer os seguintes cenários.

- O BOT não consegue concluir o teste porque algo inesperado aconteceu, como, por exemplo, não ser encontrada nenhuma janela com um determinado título.
  Nesse caso, é tirado um *screenshot* e é escrito no relatório gerado o erro obtido ou qual a parte do teste em que ocorreu o erro.

- O BOT conseguiu executar o teste, mas o resultado final foi inesperado.
  Nesse caso, também é tirado um *screenshot* e é indicado no relatório que o teste falhou.
  Caso seja aplicável e faça sentido para o teste, gera também um ficheiro txt com as diferenças entre o teste atual e um teste executado anteriormente (antes e depois).

- O BOT conseguiu executar o teste e o resultado final foi o esperado.
  Nesse caso, é tirado um *screenshot* como prova e no relatório indica que o teste foi concluído com sucesso.

Finalmente, é gerado um report.html que contém, para cada teste, se o teste foi executado com sucesso ou não e o *PrintScreen* "antes"e "depois"para cada teste (figura 13).

Um teste é considerado:

- "Antes"caso não exista nenhum teste na diretoria `output` para essa transação.

- "Depois"caso já exista esse teste na directoria `output`. Nesse caso, o "antes"pode ser usado para verificar se o resultado foi o mesmo que o depois.

3.4.1 Testes Realizados

O BOT é capaz de efetuar os seguintes testes no SAP:

- Exibição de dados de maestros

- Criar um registo

- Modificar um registo
- Exibir estrutura Organizativa
- Visualização da informação geral da estrutura organizativa
- Avaliação de Tempos
- Avaliação do tempo acumulado mensal
- Consultar calendário de feriados
- Executar folha de pagamentos
- Consultar esquema de tempos
- Execução de um relatório SAP padrão
- Programa Preliminar SEPA
- Contabilização
- Validação de documentos contabilísticos
- Consultar o uso do conceito de folha de pagamento
- Executar um relatório legal
- Executar um relatório Z (Custom)
- Testar conexões
- Teste de bloqueio de correio de saída
- Testes de acesso em utilizadores de teste
- Serviços SICF: verificar se os serviços devem ser ativados
- Verificar as configurações dos e-mails
- Verificar a ativação dos serviços de fluxo de trabalho

E os seguintes testes no Portal Global:

- Auto-serviço do empregado
- Gestor de Auto-serviço e Planeador de Turno

Nos apêndices A encontram-se uma tabela com uma breve descrição e o resultado esperado para cada teste.

Em baixo encontram-se alguns testes de forma mais detalhada, constando como foram implementados, e nos apêndices B constam os *screenshots* de alguns testes efetuados.

3.4.1.1 Criação registo

Para efetuar a Criação de um Registro é necessário um ficheiro CSV com uma lista de números de empregados, e o número de infotipo e subtipo e o intervalo da data, desde e até do registo que é para criar (figura 6).

Empleado	Infotipo	Subtipo	Desde	Até
21000233	2001	0130	11.01.2022	11.01.2022
00000022	2001	0130	11.01.2022	11.01.2022
00000022	2001	0130	11.01.2022	11.01.2022

Figura 6: Folha de excel com a lista de dados a serem testados. Neste exemplo, o 3º teste irá dar erro porque vai tentar inserir dados que já foram inseridos pelo 2º teste.

O teste começa por inicializar a transação pa30 e verifica se o título da janela é *Actualizar datos maestros personal*. Se não for, significa que a transação não foi inicializada corretamente. Nesse caso, é indicado no relatório que a transação não foi efetuada.

Depois, são usados os valores lidos da folha de excel referida acima, com a exceção das datas que vão ser lidas posteriormente. O BOT envia o valor das variáveis da folha de excel para as caixas de texto corretas (figura 7).

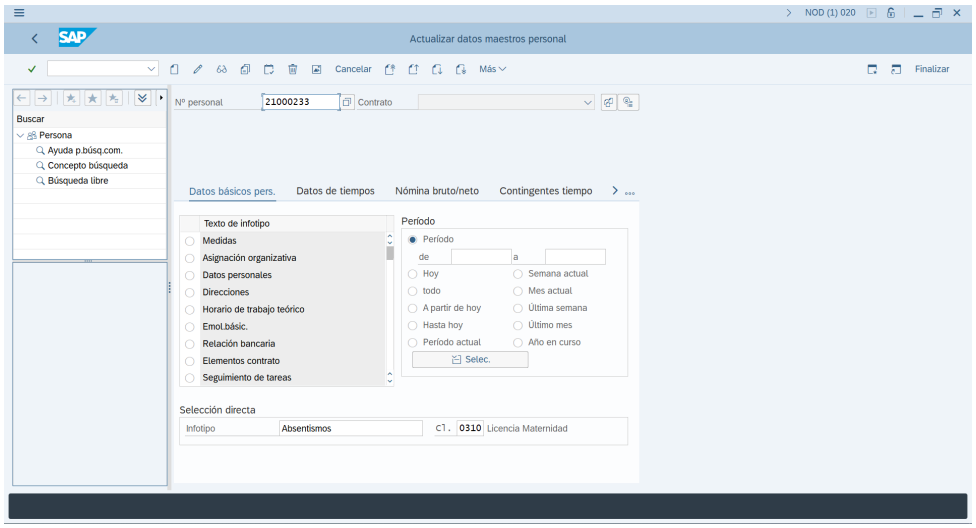


Figura 7: Criação de um registo ausências devido a licenças de maternidade para um determinado empregado

A seguir é clicado no botão para criar o registo. Neste momento, a janela continua com a mesma barra de título, mas o conteúdo deve ter mudado. Se o texto em foco continuar a ser o mesmo significa

que ocorreu um erro, como, por exemplo, o número de empregado não existe (figura 8). Tal como todos os casos de erro, é lido a mensagem de erro existente na barra de estado, utilizando o *SAP GUI Scripting API*, e é tirado um screenshot da janela para colocar no relatório.

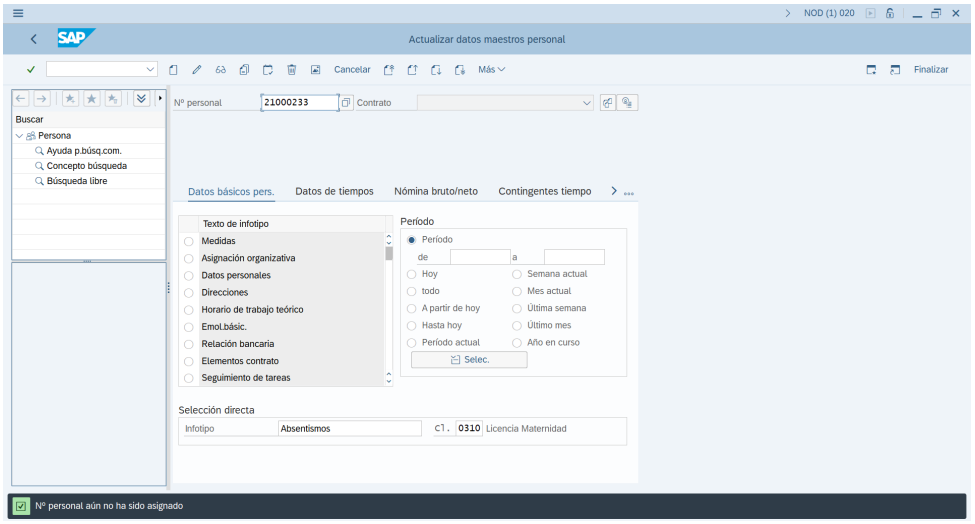


Figura 8: Não foi possível criar o registo porque o empregado não existe

Se não ocorreu nenhum erro, são colocadas as datas "desde"e "até"(figura 11).

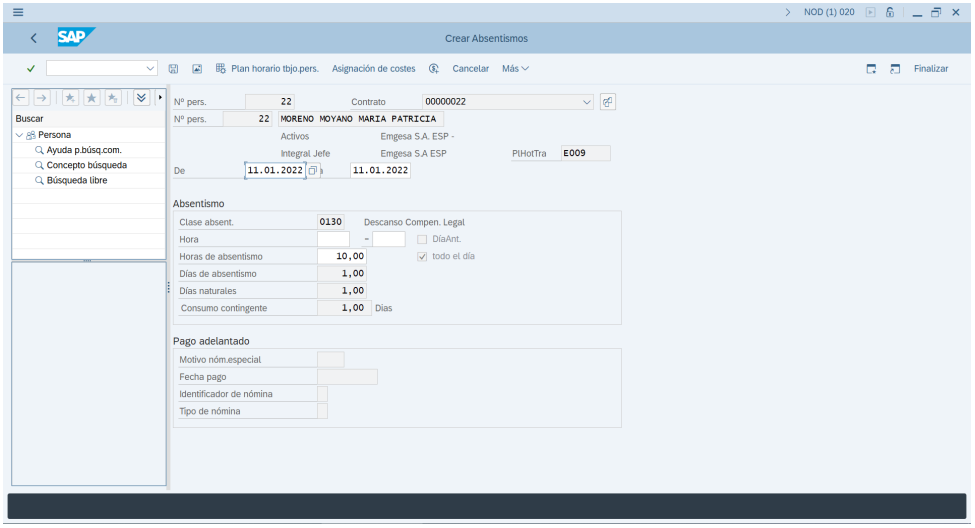


Figura 9: Escolha do intervalo durante a criação do registo

Após ter inserido as datas, da mesma forma que foram inseridos os outros valores, é gravado o registo.

Se ocorreu algum erro, por exemplo, se já houver um registo igual (figura 10), é tirado um *screenshot* à janela que contém o erro e é inserido no relatório gerado.

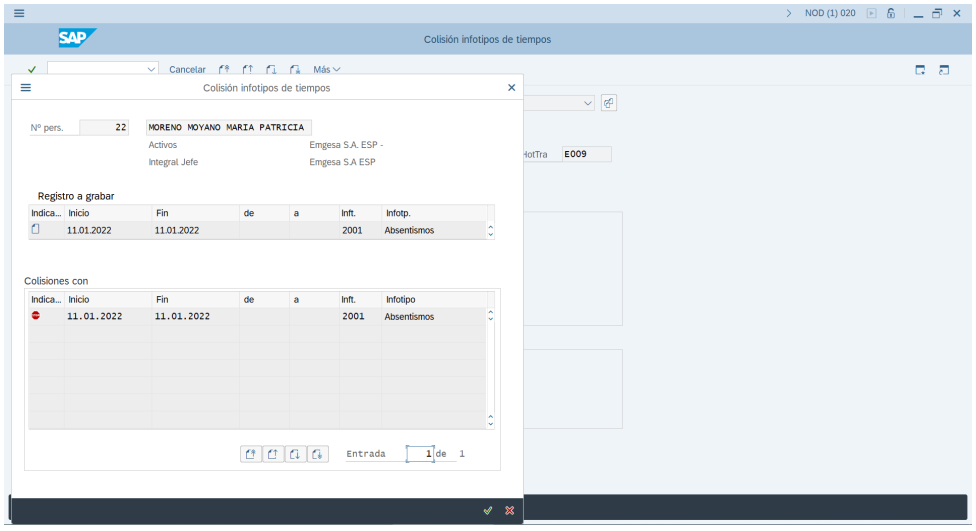


Figura 10: Escolha do intervalo durante a criação do registo

Se o SAP HCM não retornou nenhum erro (figura 11, é ainda necessário verificar se o registo foi de facto criado.

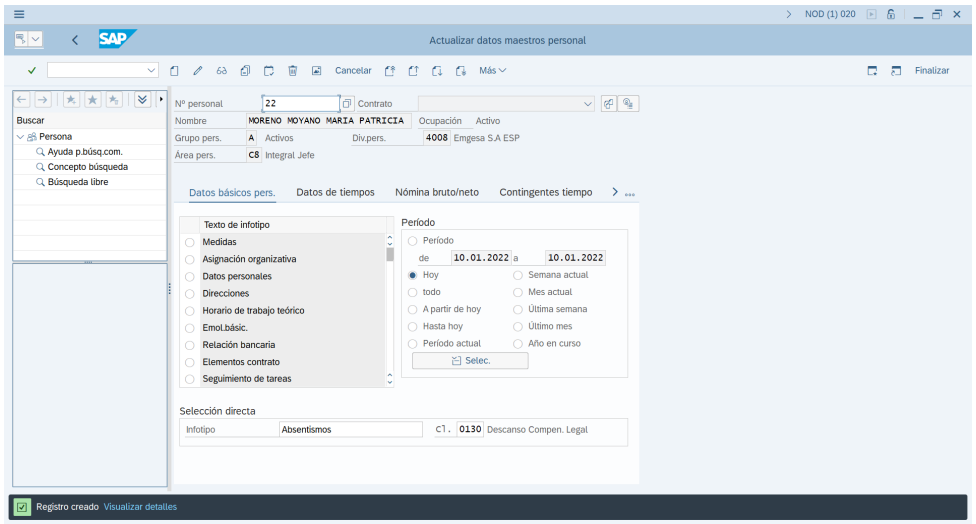


Figura 11: Tela do SAP após a criação de um registo

Para isso, é utilizada a funcionalidade do SAP para apresentar um sumário dos registos criados. São novamente inseridas as datas "desde"e "até"para reduzir o número de resultados a serem apresentados.

Agora é apresentada uma tabela com todos os registos para as datas selecionadas (figura 12. É lida cada linha da tabela com o intuito de encontrar o registo que acabou de ser criado.

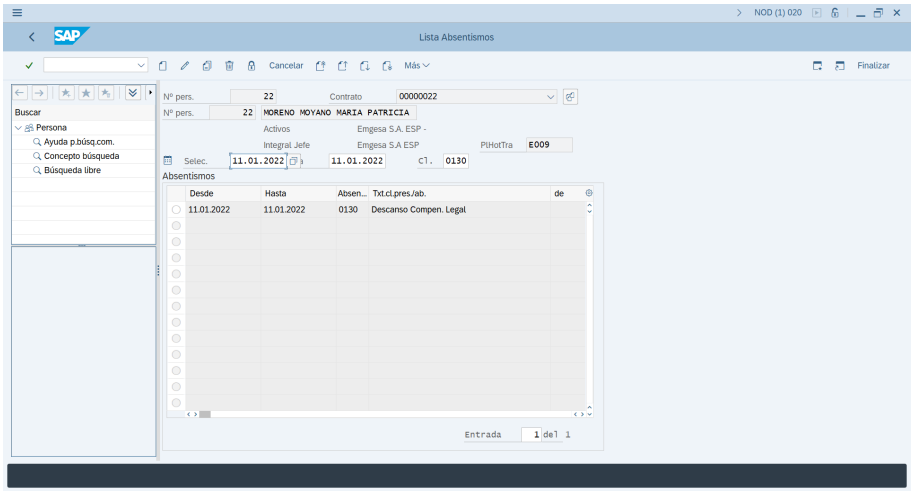


Figura 12: Por último é verificado se o registo foi criado

Dependendo se o registo foi encontrado, o teste foi ou não concluído com sucesso.

Por último, é repetido o processo para outros registos, caso existam.

Quando concluído, é gerado o seguinte relatório:

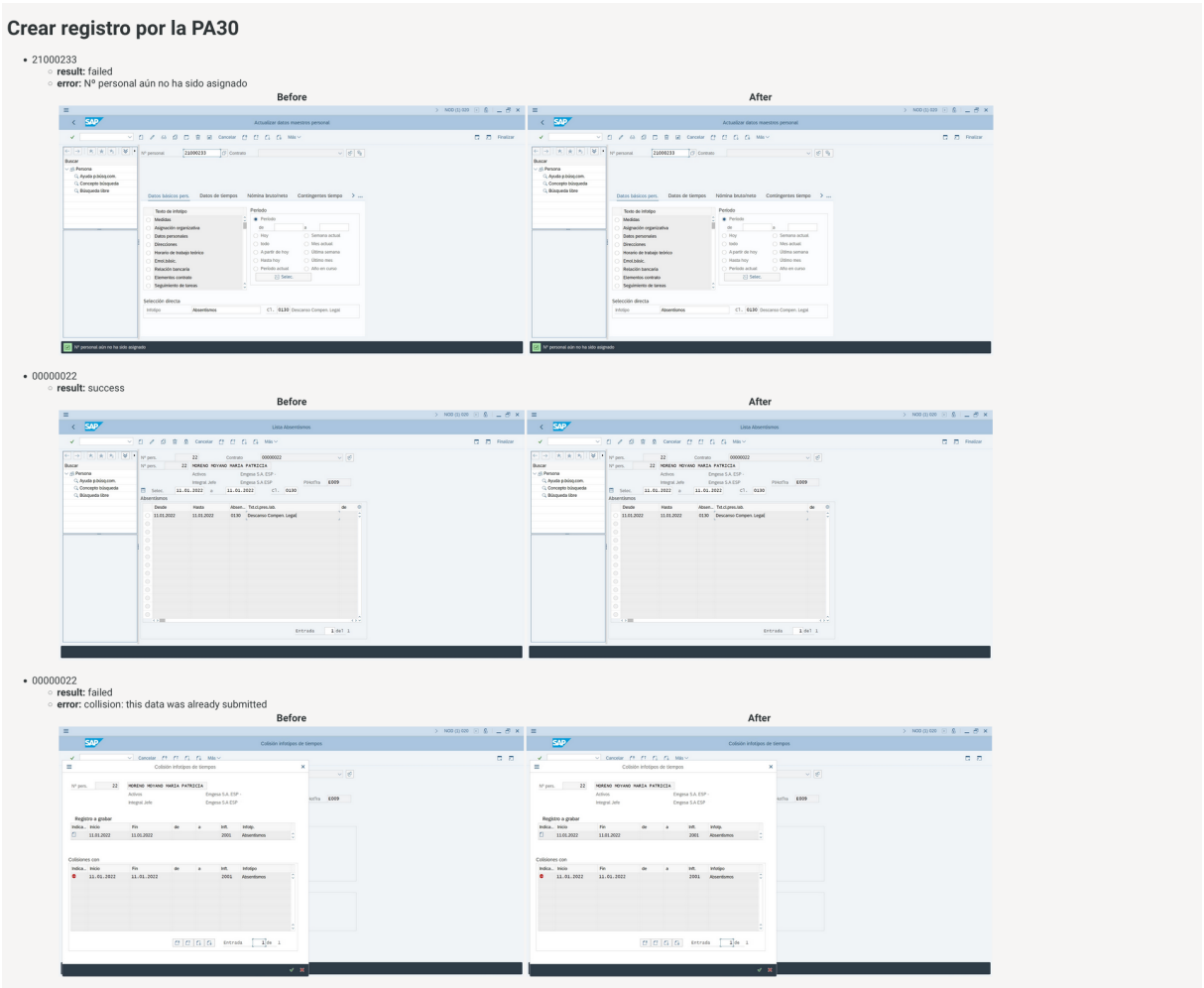


Figura 13: *Printscreen* do relatório gerado

3.4.1.2 Modificar Registo:

Este teste altera um registo existente para uma nova data.

Para além das variáveis existentes na folha de excel do teste criar registo, também contém uma nova data (figura 14).

Empleado	Infotipo	Subtipo	Desde	Até	NewStart	NewToTo
22	2001	0130	11.01.2022	11.01.2022	12.01.2022	12.01.2022
22	2001	0130	12.01.2022	12.01.2022	11.01.2022	11.01.2022

Figura 14: Folha de excel com a lista de dados a serem testados

Modificar um Registo criado é muito semelhante à Criação de um Registo. Usa a transação pa61, que, apesar de não ser a mesma que a criação do registo, a interface é idêntica o que significa que o BOT pôde usar as mesmas funções para interagir com a janela (figura 15).

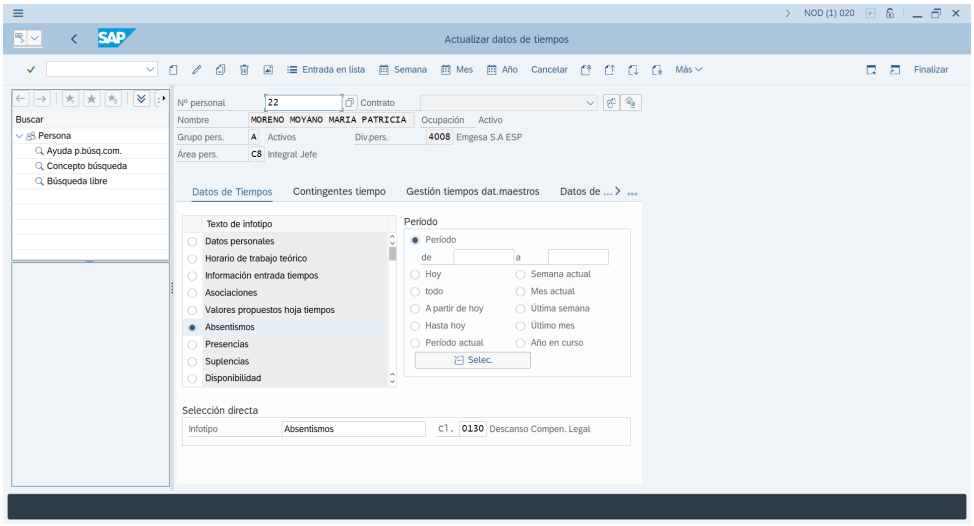


Figura 15: Modificação de um registo existente

Depois de inserir os valores, é clicado no botão para modificar o registo (figura 16).

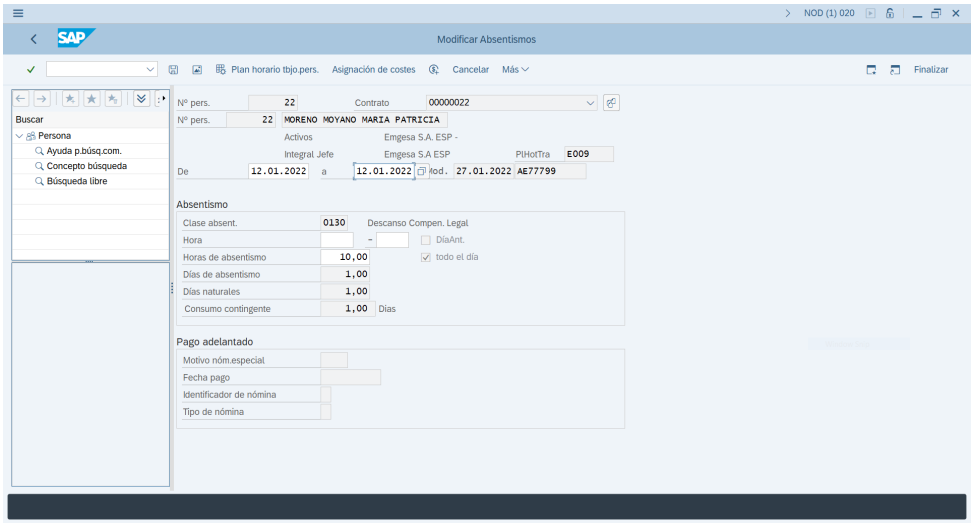


Figura 16: Momento em que a data é alterada

Por último, se não ocorreu nenhum erro ao modificar o registo, o BOT verifica se a data foi de facto alterada (figura 17).

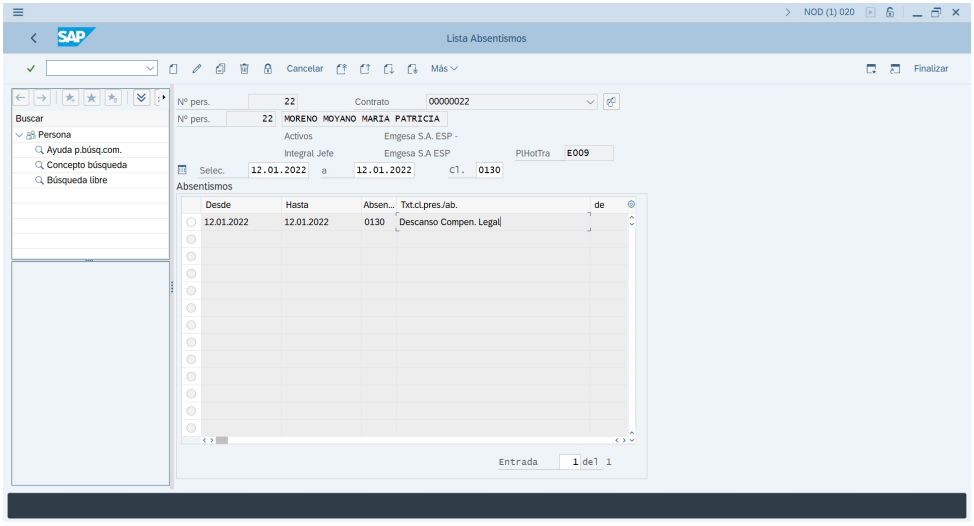


Figura 17: Prova que a data foi alterado com sucesso

3.4.1.3 Avaliação de Tempos

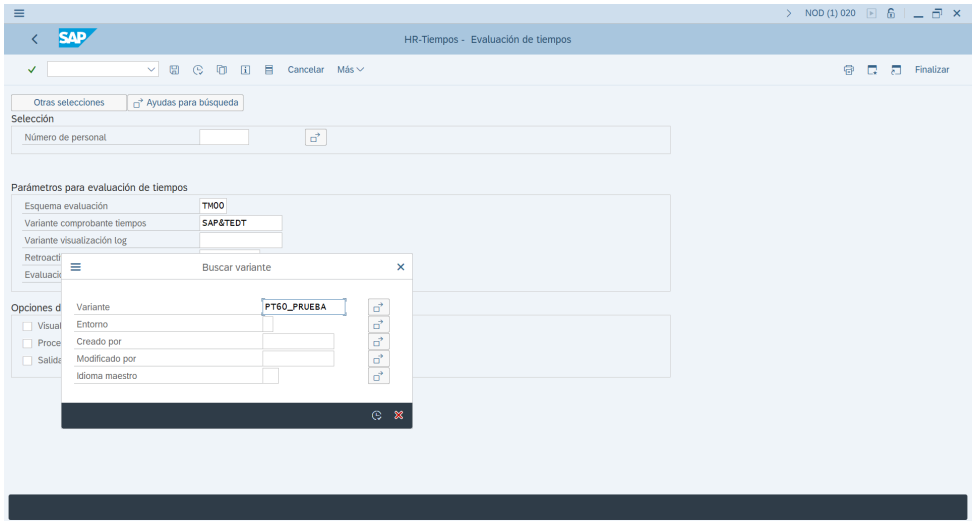
Este teste avalia se é executada corretamente a avaliação dos tempos.

A *spreasheet* apenas contém uma lista de variantes a serem usados pelo teste (figura 18).

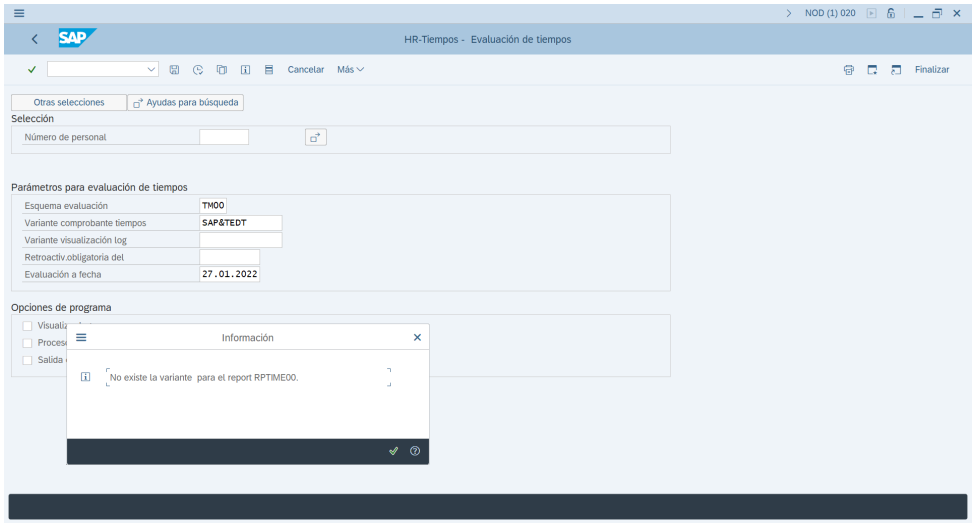
	A
1	Variant
2	PT60_PRUEBA
3	ZPRUEBA

Figura 18: Folha de excel com a lista de dados a serem testados

Depois de iniciar a transação pt60, é pesquisado pelo 1º variante que está na *spreasheet* e retorna um erro caso o variante não exista (figura 19b) e é adicionado um *screenshot* ao relatório com a mensagem de erro.



(a) Selecionar variante



(b) Variante não encontrada

Figura 19: Inserir variante presente no ficheiro csv

Após executar o teste com a variante (figura 20), é exportado o resultado para um ficheiro que vai

ser comparado e, caso difira, é criado outro ficheiro com as diferenças usando o comando FC existente no Windows.

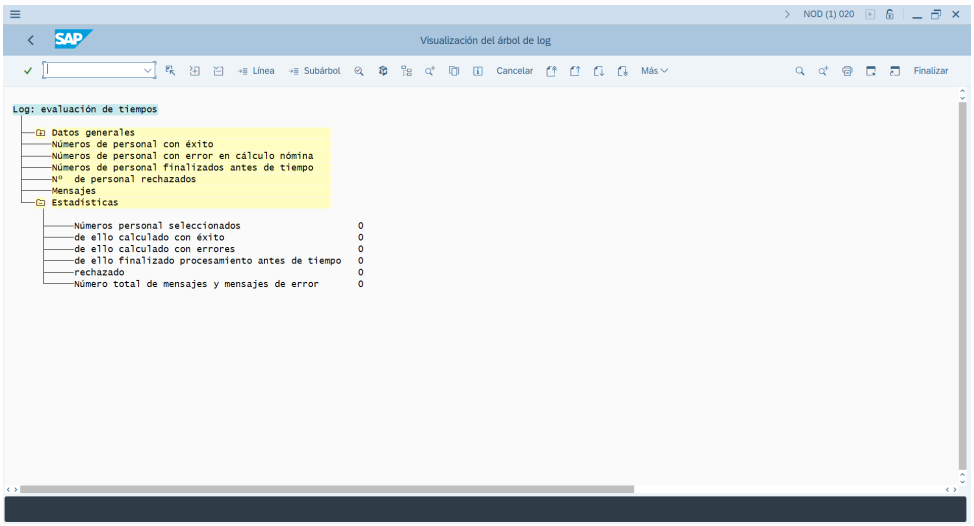


Figura 20: Resultado obtido ao executar a variante

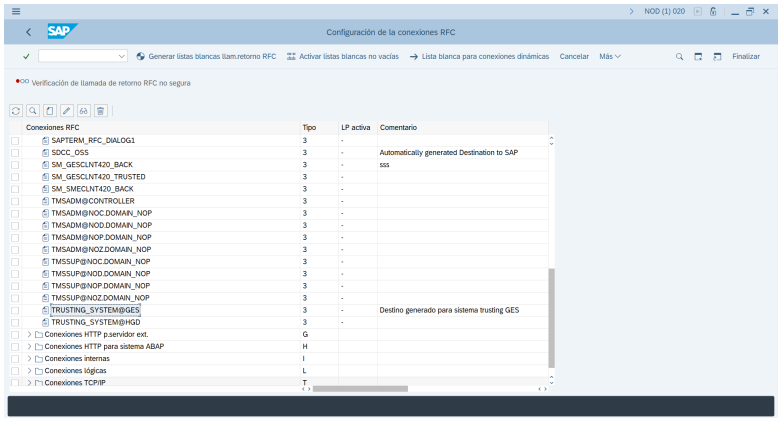
3.4.1.4 Testar conexões SM59

Nesta transação, é necessário uma *spreadsheet* que contenha uma lista de conexões que irão ser testadas (figura 21)

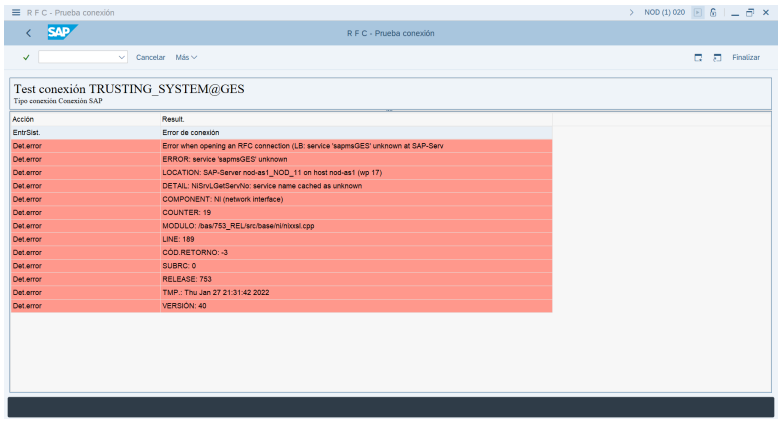
	A	B
1	Conexíons	Tipo
2	CO01D4P200	3
3	KLS	3
4	BWDB35020	3
5	KLS	3
6	TRUSTING_SYSTEM@GES	3
7	CSI_AWS_EC2	G

Figura 21: Folha de excel com a lista de conexões a serem testadas

É executado a transação SM59 e são testadas todas as conexões presentes na *spreadsheet* (figura 22).



(a) Selecionar conexão



(b) Erro na conexão

Figura 22: Testar conexões na transação SM59

3.4.1.5 Portal Global - Verificar se todas as opções funcionam - Auto-serviço do empregado

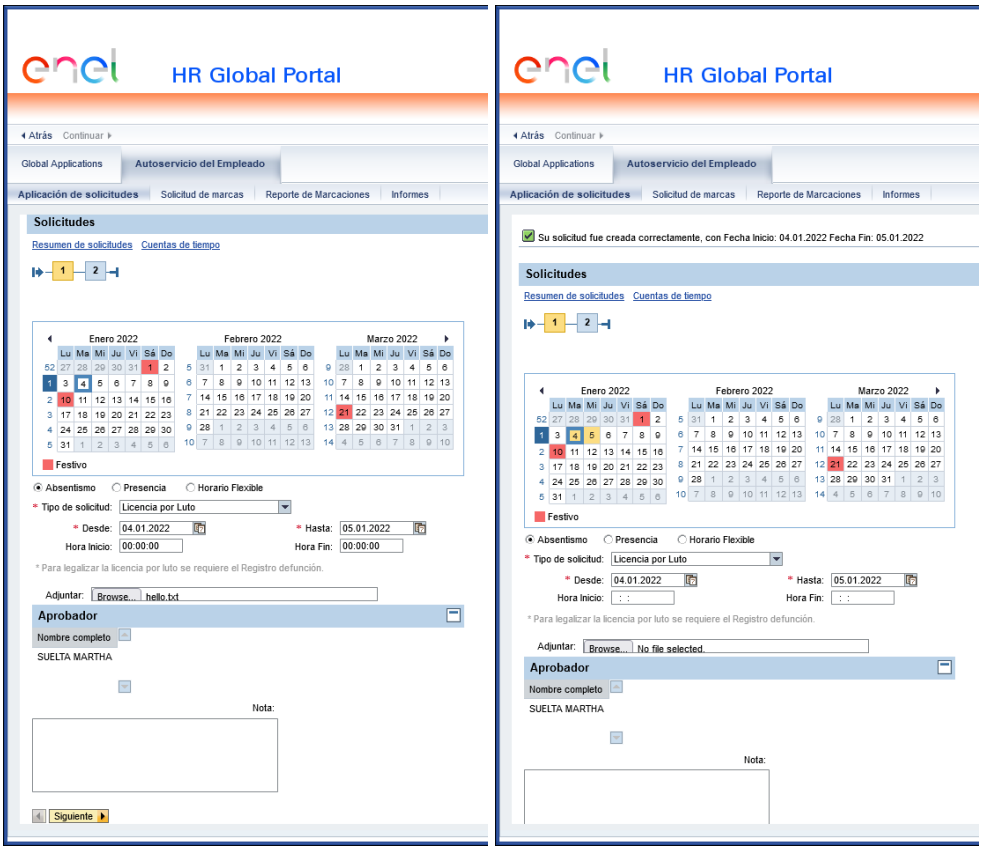
O BOT começa por iniciar sessão no Portal Web utilizando o utilizador de Empregado (figura 23).

Figura 23: Página de Login do Portal

De seguida, o BOT vai simular que o empregado quer efetuar pedidos de marcações.

Para isso, o BOT começa por clicar no separador "Autoservicio del Empleado"e depois "Aplicación de solicitudes".

Depois é selecionado o tipo de pedido (licença por luto) e o intervalo de data e é anexado um ficheiro (figura 24a).



(a) Criação de um pedido de marcação (b) Mensagem após concluir pedido

Depois de criar o pedido, é verificada se a frase "Su solicitud fue creada correctamente"existe na página (figura 24b).

Se existir, é necessário a aprovação do *manager*. Por essa razão, agora é novamente efetuado um *login*, mas desta vez é utilizado o utilizador do *manager* (figura 23).

Depois de efetuar o *login*, o BOT clica no botão com o texto "Autoservicio del Gestor".

Por último, procura na tabela, linha a linha, pelo pedido criado previamente, e tenta aprová-lo (figura 25)

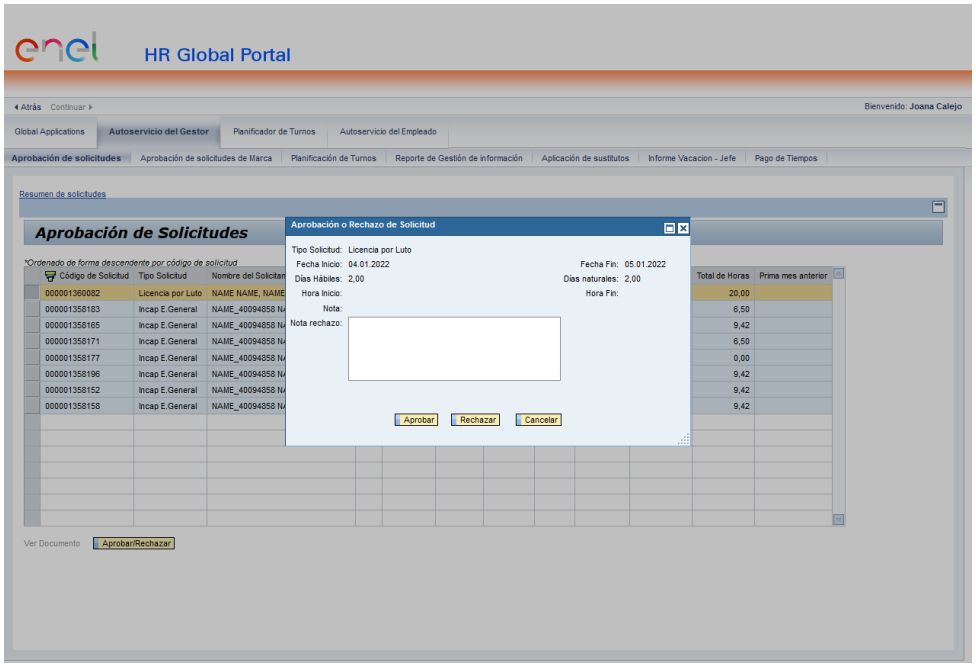


Figura 25: Janela para aprovar ou rejeitar o pedido de marcação

3.5 Manutenção do BOT Futura

Tal como foi dito em 2.3, de modo a automação ser eficiente e útil, não deve ser despendido muito tempo na criação e manutenção dos testes automáticos.

Caso se pretenda criar um teste, é preciso criar a estrutura como é referido em 3.4. Por exemplo:

```
NovoTeste := {enabled: false ; Desativado por omissão
             ,title: "Novo teste" ; Nome do teste
             ,hasData: true ; Tem um ficheiro CSV com variáveis associadas
             ,type: "PORTAL"} ; É executado no portal
```

Na criação do BOT, houve o cuidado de criar funções que são comuns para vários os testes, tais como a seleção da variante, exportação de uma tabela para um ficheiro ou a comparação de 2 ficheiros. Isto implica que, caso algo seja alterado numa atualização do SAP ou exista algum *bug* no BOT, apenas seja necessário alterar essas funções 1 vez.

Outro facto que reduz o tempo de manutenção do BOT, é o uso do *SAP GUI Scripting*. O SAP permite gravar os passos executados manualmente para um script para um ficheiro escrito em Microsoft SBscript (.vbs). Após gerar o script, é possível copiar para o código do BOT, e, se necessário, substituir as variáveis *hard coded* pelas variáveis existentes no ficheiro CSV.

Por exemplo, a linha de código gerada pelo SAP:

```
session.findById("wnd[0]/usr/ctxtRS38M-PROGRAMM").text = "Texto a ser
↪  adicionado"
```

Ao adicionar ao BOT, ficaria:

```
session.findById("wnd[0]/usr/ctxtRS38M-PROGRAMM").text := data[1]
```

Em que `data[1]` corresponde à 1° coluna do ficheiro CSV

3.6 Conclusão

Finalizada a dissertação, foi cumprido o objetivo de desenvolver uma ferramenta BOT capaz de realizar vários testes e operações, de forma automática.

Para tal, desenvolvi um BOT escrito em AutoHotKey e partes em Python que executa os testes, permitindo que não houvesse necessidade de os fazer manualmente, como até então.

Cada teste tem o resultado "antes"e "depois", ou seja, a 1ª vez que o teste seja executado, é considerado o teste "antes"e pode ser usado como referência para avaliar o teste "depois". Se o resultado "depois"diferir do "antes", é criado um ficheiro com as diferenças.

Além disso, o BOT também gera um relatório para cada teste no formato `.html`, que contém, para cada teste, o resultado do teste (passou, não passou ou desconhecido), uma descrição (por exemplo, porque é que o teste falhou) e screenshots dos resultados "antes"e "depois".

Adicionalmente, a automação é *Data Driven*: o BOT suporta diferentes condições de teste e variáveis carregadas de um ficheiro CSV associado ao teste.

Como trabalho futuro, pode, por exemplo, ser melhorada a aparência do template do relatório, adicionando CSS ao código HTML gerado pelo BOT, ou criando uma ferramenta que ajuda a implementar novos testes, por exemplo, gerar um *template* de código com o nome do teste. Além disso, em vez de usar um ficheiro CSV com os dados a serem testados, o BOT poderia gerar os inputs de forma aleatória.

Bibliografia

[1] AutoHotkey. url: https://github.com/Lexikos/AutoHotkey_L/blob/master/license.txt (ver p. 13).

[2] AutoHotkey. *CoordMode - Syntax & Usage | AutoHotkey*. url: https://www.autohotkey.com/docs/commands/CoordMode.htm (ver p. 14).

[3] A. Axelrod. *Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects*. Apress, 2018. isbn: 9781484238318, 9781484238325 (ver p. 1).

[4] Cloudflare. *What is a bot? | Bot definition*. http://web.archive.org/web/20220521042015/https://www.cloudflare.com/learning/bots/what-is-a-bot/ (ver p. xii).

[5] J. P. Elfriede Dustin Jeff Rashka. *Automated Software Testing: Introduction, Management, and Performance*. Addison-Wesley Professional, 1999. isbn: 0201432870,9780201432879 (ver p. 1).

[6] P. Ercoli e F. Bauer. «Software Engineering Techniques». Em: *Rome, Italy, 27th To 31st October 1969, http://homepages. cs. ncl. ac. uk/brian. randell/NATO/n ato1969. PDF (found 20/06/2010 09: 45)* (1969) (ver p. 2).

[7] J. M. Lourenço. *The NOVAthesis LaTeX Template User's Manual*. NOVA University Lisbon. 2021. url: https://github.com/joaomlourenco/novathesis/raw/master/template.pdf (ver p. ii).

[8] D. G. Mark Fewster. *Software test automation: effective use of test execution tools*. illustrated edition. Addison-Wesley, 1999. isbn: 9780201331400,0-201-33140-3. url: http://gen.lib.rus.ec/book/index.php?md5=fe69d82b2ff7118a27e6070cc9e5937f (ver p. 9).

[9] Microsoft. *Lifecycle FAQ - Internet Explorer and Microsoft Edge*. url: https://docs.microsoft.com/en-us/lifecycle/faq/internet-explorer-microsoft-edge (ver pp. 15, 16).

[10] V. Paradigm. *What is User Story?* http://web.archive.org/web/20210910181535/https://www.visual-paradigm.com/guide/agile-software-development/what-is-user-story/ (ver p. xii).

[11] SAP. *SAP GUI Scripting - SAP Help Portal*. url: https://help.sap.com/viewer/b47d018c3b9b45e897faf66a6c0885a8/760.03/en-US (ver p. 14).

[12] Selenium. *Selenium | Getting started*. url: https://www.selenium.dev/documentation/webdriver/getting_started/ (ver p. 15).

[13] I. Sommerville. *Software Engineering, 10th Edition*. 10ª ed. Global Edition. Pearson, 2016. isbn: 9781292096131 (ver p. 2).

[14] G. Trends. url: https://trends.google.com/trends/explore?date=all&q=%2Fm%2F0555tg,%2Fm%2F070m83 (ver p. 13).

[15] K. L. M. Wu. *Effective GUI test automation : developing an automated GUI testing tool*. SYBEX, 2005. isbn: 1417554835,9781417554836 (ver p. 10).

A

Lista de testes

Teste	Descrição	Resultado
Exibição de dados de mestres	Usar a transação SE16, tabela PA0002, para apresentar uma lista com todos os funcionários existentes numa determinada empresa, como por exemplo nome, apelido, morada, etc. Esta lista é exportada para um ficheiro de texto	É exportada a lista de registos para um ficheiro de texto. O teste passou se conseguiu exportar e se o registo exportado "depois"for igual ao "antes". Caso sejam diferentes, é criado um ficheiro novo com as diferenças.
Criar um registo	Criar um registo novo na transação PA30 como, por exemplo, para criar uma ausência de maternidade a um trabalhador.	É criado o registo de ausência. Se não ocorrer nenhum erro, é verificado se o registo criado se encontra na lista de registos. Caso se encontre, então passou o teste.
Modificar um registo	Modifica um registo na transação PA61 para uma nova data. Por exemplo, alterar a ausência de maternidade criada no teste anterior.	Se não ocorreu nenhum erro e se o registo se encontra modificado na lista de registos, então passou o teste.
Exibir estrutura Organizativa	Utiliza a transação PPOSE para apresentar todos os trabalhadores pertencentes uma unidade organizacional (departamento da empresa).	É considerado que o teste passou se o resultado for igual ao teste "antes", ou seja, contém todos os trabalhadores da unidade organizacional.

Continua na página seguinte

Tabela 3 – continuação da página anterior

Teste	Descrição	Resultado
Visualização da informação geral da estrutura organizativa	Utiliza a transação PP01 para apresentar a informação sobre uma determinada unidade organizacional.	Se lista exportada for igual à do teste "antes", então passou o teste.
Avaliação de Tempos	Utiliza a transação PT60 para efetuar o processamento de ausências e presenças para um empregado	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é verificado se foi executado sem erros.
Avaliação do tempo acumulado mensal	Executar a transação PT_BAL00 para apresentar, por exemplo o tempo de ausências mensal para um determinado empregado	O resultado é exportado para um ficheiro. Se o teste anterior for igual, então passou o teste.
Consultar calendário de feriados	Utilização da transação SCAL para apresentar calendário de feriados e consequente lista de feriados para um determinado País.	Exporta a lista de feriados e compara com os feirados do teste "antes". Se a lista de feriados for igual, então é considerado que o teste passou.
Executar folha de pagamentos	Utiliza a transação pc00_m38_calv para visualizar o recibo de pagamento dos funcionários	Exporta o log obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.
Consultar esquemas de tempos	Utiliza a transação SE38, para um dado programa e esquemas de tempos que permite visualizar as regras cálculo definidas para determinada empresa.	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.
Execução de um relatório SAP padrão	Utiliza a transação pc00_m99_cwtr para visualização de rubricas salariais	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.
Programa Preliminar SEPA	Utiliza a transação pc00_m99_cdta para visualizar as transferências bancárias para um determinado empregado	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.

Continua na página seguinte

Tabela 3 – continuação da página anterior

Teste	Descrição	Resultado
Contabilização	Utiliza a transação PC00_M99_CIPE para visualizar a preparação do pagamento salarial	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.
Validação de documentos contabilísticos	Utiliza a transação PCP0 e permite ver os detalhes dos cálculos contabilísticos realizados.	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.
Consultar o uso do conceito de folha de pagamento	Utiliza a transação pc00_m99_dlga20 para ver as propriedades das rubricas salariais, como por exemplo para que empregado é calculada e que cálculos está associada.	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.
Executar um relatório legal	Utiliza a transação pc00_m38_fie para visualizar os empregados ativos na empresa.	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.
Executar um relatório Z (*Custom*)	Utiliza a transação zhr_analisis_wids permite visualizar os *work items* que correspondem a cada tarefa de fluxo de trabalho (por exemplo quando um empregado faz um pedido de ausência ao seu gestor, e este aprova ou rejeita).	Exporta o resultado obtido pelo SAP para um ficheiro. De seguida é comparado com o teste anterior e se for igual, então passou o teste.
Testar conexões	Utiliza a transação SM59 para testar a conexão remota entre o sistema SAP ERP e outro sistema externo.	Considera que o teste passou ou falhou dependendo do resultado das conexões (”Connection OK”ou ”Connection Error”).
Teste de bloqueio de correio de saída	Utiliza a transação SOST para visualização dos *e-mails* que ainda não foram enviados e tenta abrir um deles	Passou o teste se não ocorreu nenhum erro.

Continua na página seguinte

Tabela 3 – continuação da página anterior

Teste	Descrição	Resultado
Testes de acesso em utilizadores de teste	Utiliza a transação SU01 para verificar se autorizações do utilizador não foram alteradas.	Exporta a tabela, remove linhas vazias e compara com o teste "antes". Se estiver igual, então o teste passou.
Serviços SICF: verificar se os serviços devem ser ativados	Utiliza a transação SE38 e o programa RS_ICF_SERV_ADMIN_TASKS para verificar se existem serviços por ativar na transação SICF.	Exporta ficheiro CSV. Se o ficheiro for igual ao do teste "antes", o teste passou. Caso não seja, é criado um novo ficheiro com as diferenças.
Verificar as configurações dos e-mails	Utiliza a transação SCOT para apresentar as definições de SMTP, Fax, Internet e Busca(SMS)	O teste passou se não existir nenhum problema nas configurações.
Verificar a ativação dos serviços de fluxo de trabalho (*workflow*)	Executa a transação SWU3 para verificar a ativação dos serviços do *workflow*.	É considerado que o teste passou se os serviços ativados forem os mesmos que no teste "antes"
Portal Global - Auto-serviço do empregado	Testa todas as funcionalidades relacionadas com o serviço do empregado, como, por exemplo, efetuar um pedido de licença por luto e tentar aprovar com a conta do gestor.	O teste passou se executou todas as tarefas pretendidas sem erros e for encontrado uma mensagem no portal a indicar que a ação foi concluída com sucesso.
Portal Global - Gestor de Auto-serviço e Planeador de Turno	Testa todas as funcionalidades relacionadas com o serviço do gestor, como por exemplo aprovar pedidos de ausências do empregado.	O teste passou se executou todas as tarefas pretendidas sem erros e for encontrado uma mensagem no portal a indicar que a ação foi concluída com sucesso.

Tabela 3: Resumo dos testes efetuados

B

Screenshots dos testes do SAP

B.1 Exibição de dados de maestros

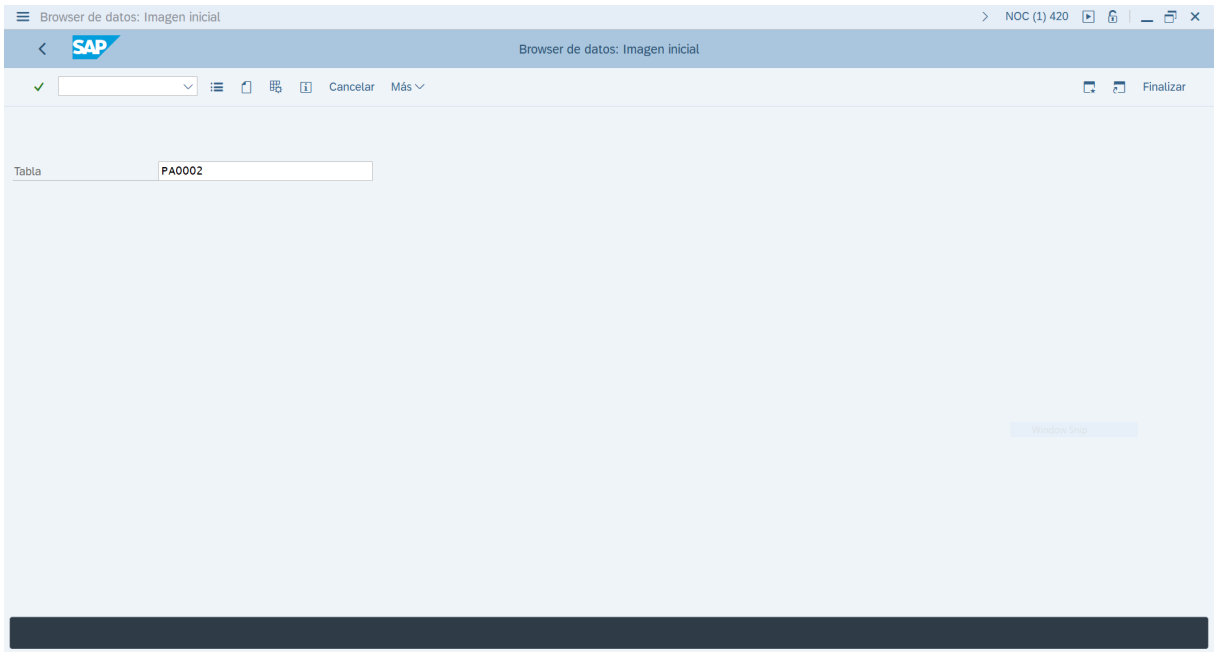


Figura 26: Selecionado tabela PA0002 na transação se16

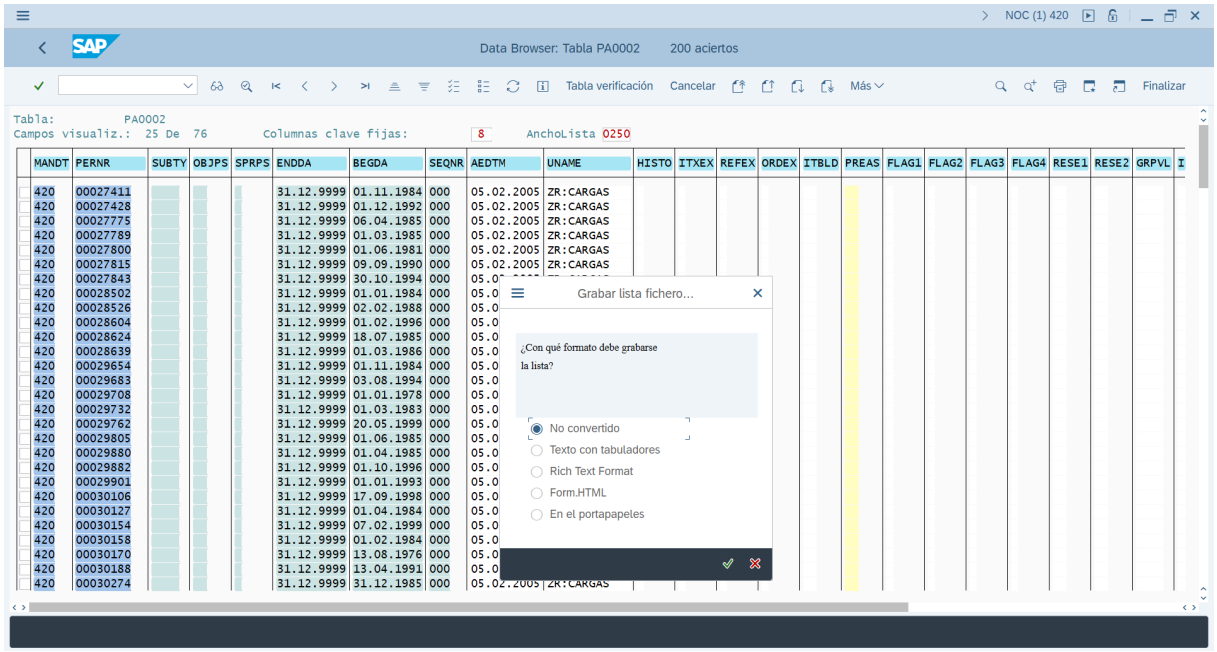


Figura 27: Os dados da tabela são exportados para um ficheiro

B.2 Avaliação de Tempos

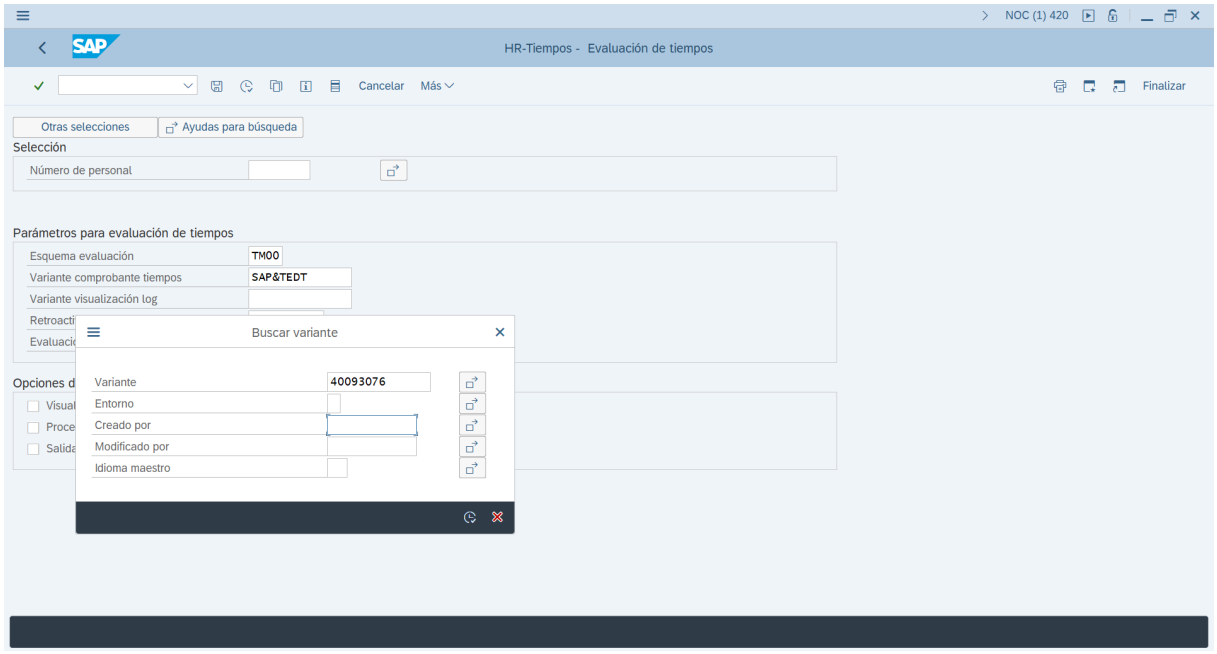


Figura 28: Escolha do variante para a transação pt60

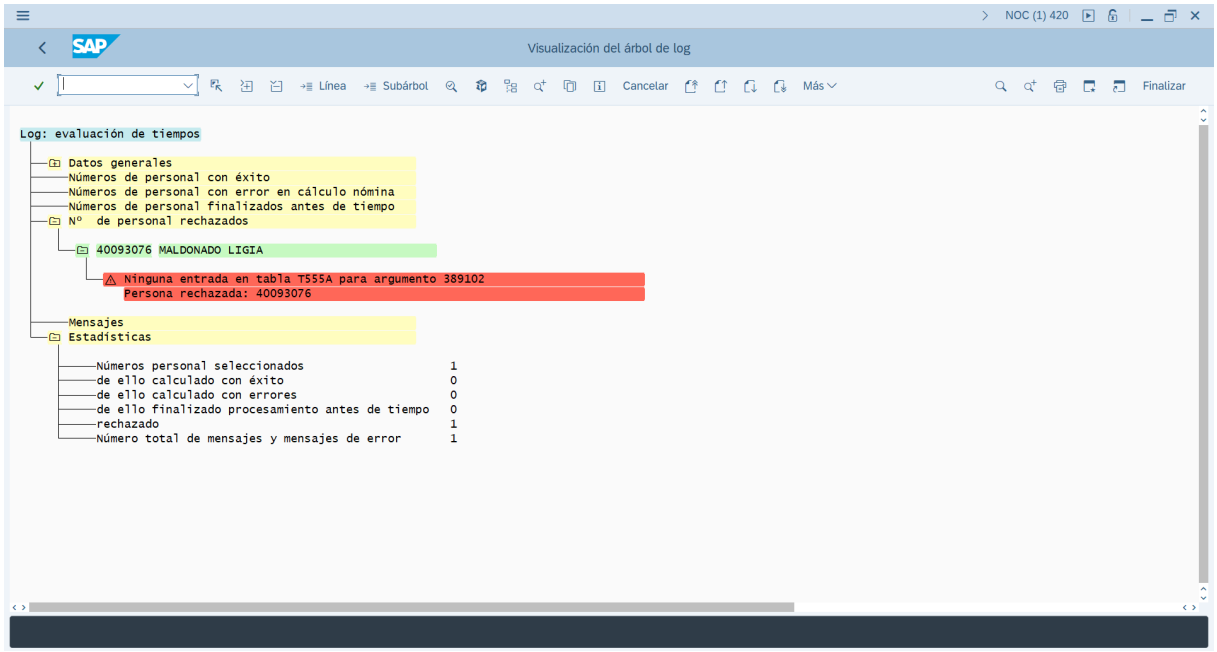


Figura 29: Output da transação pt60, com uma determinada variante selecionada

B.3 Consultar calendário de feriados

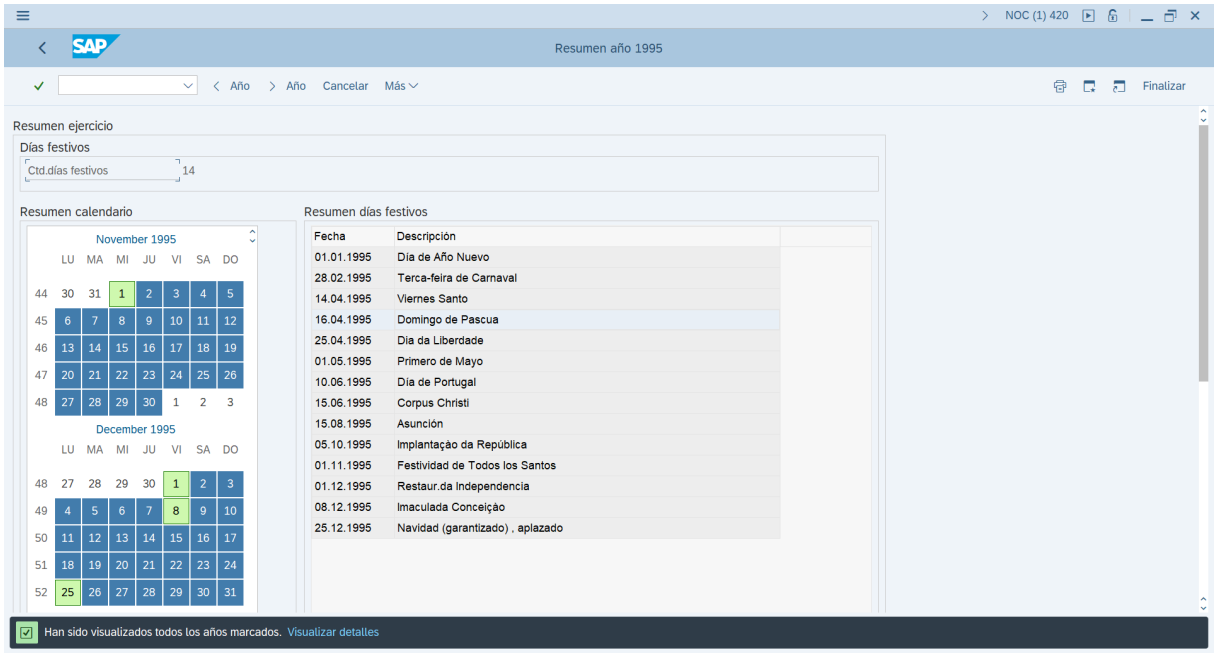


Figura 30: Lista de feriados para Portugal

B.4 Consultar esquemas de tempos

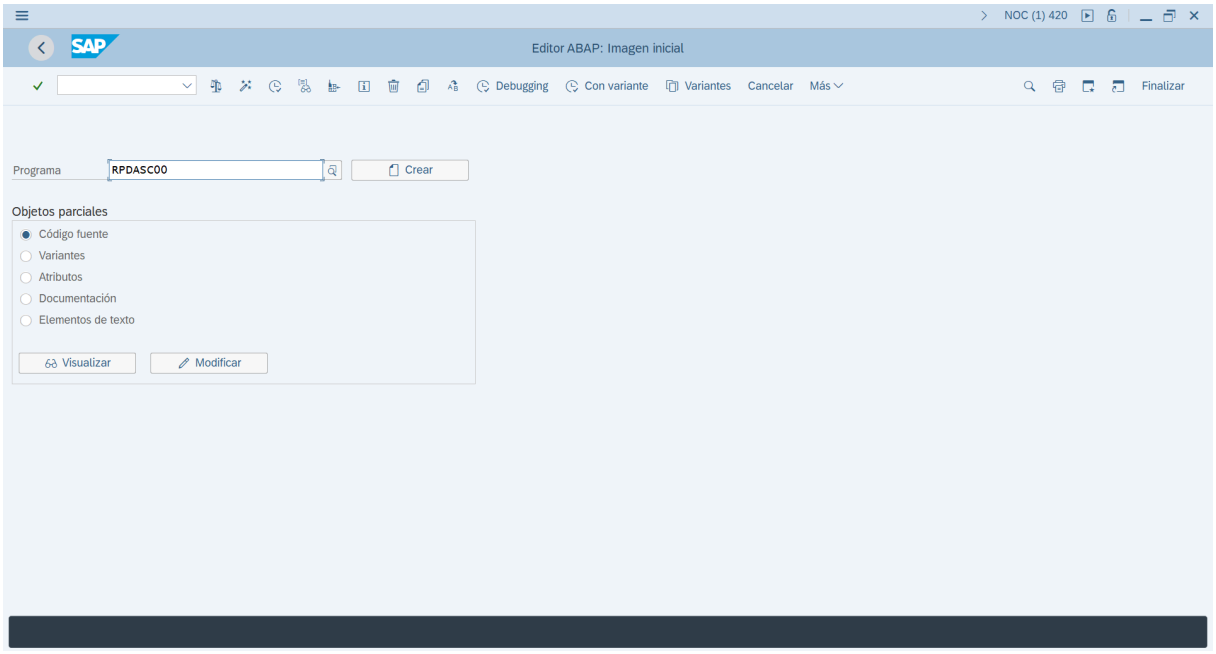


Figura 31: Executado transação SE38 e selecionado a máquina pretendida

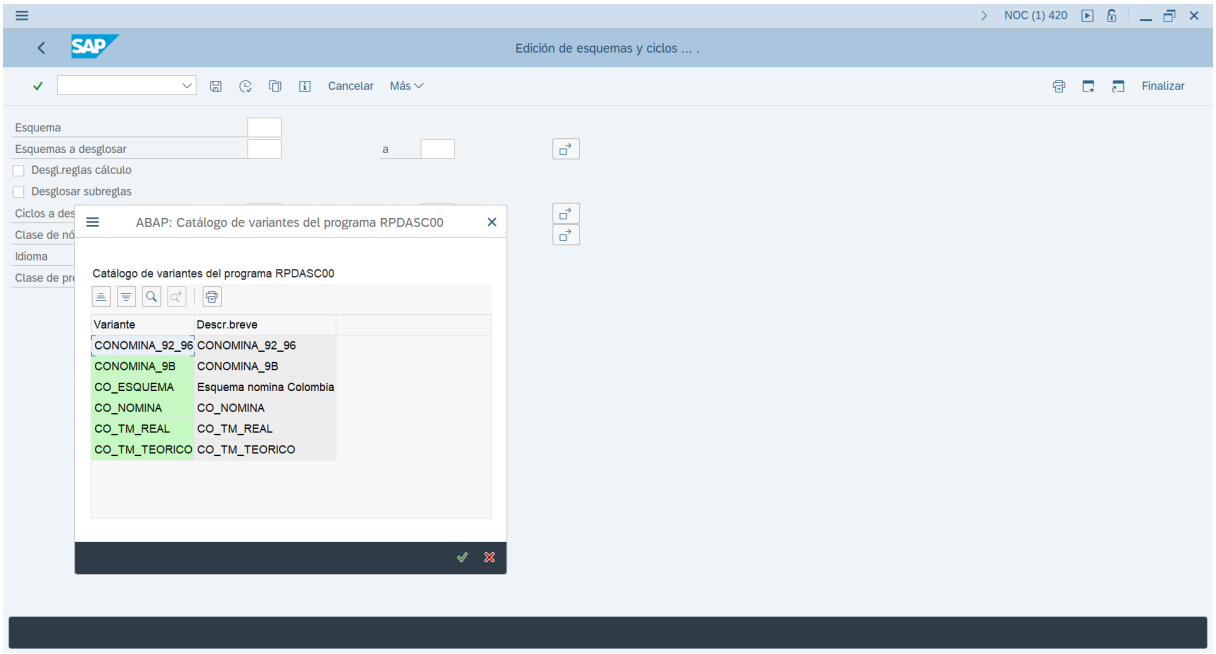


Figura 32: É selecionado o variante pretendido

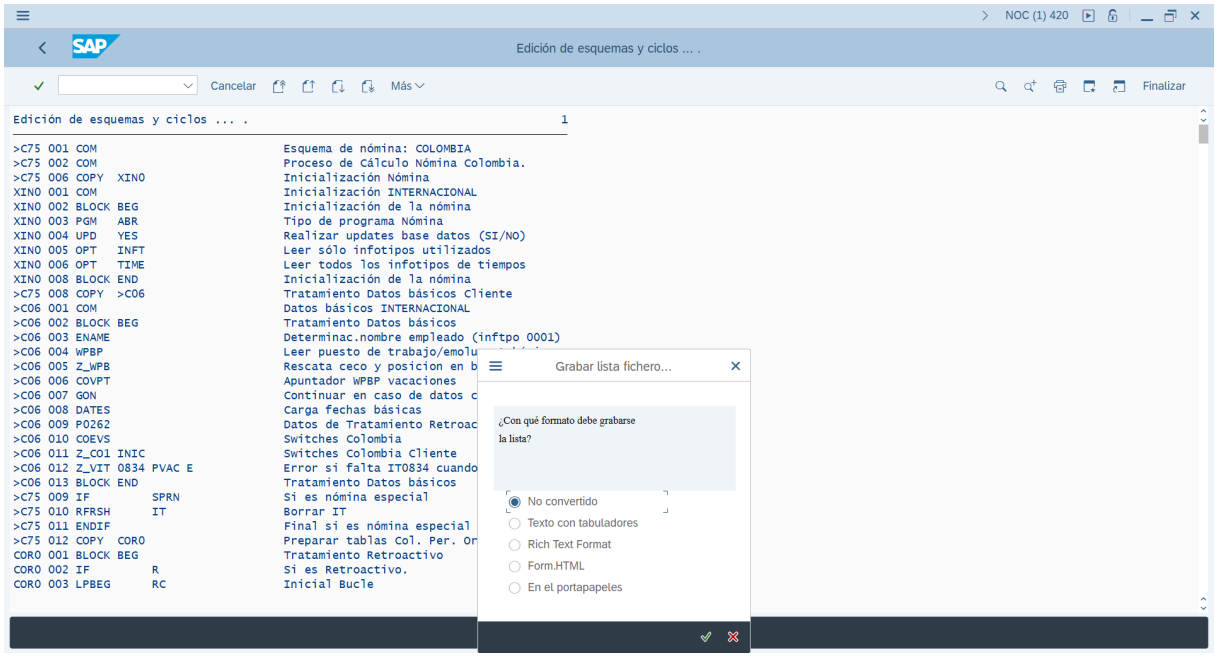


Figura 33: Exportado os resultados para um ficheiro

B.5 Validação de documentos contabilísticos

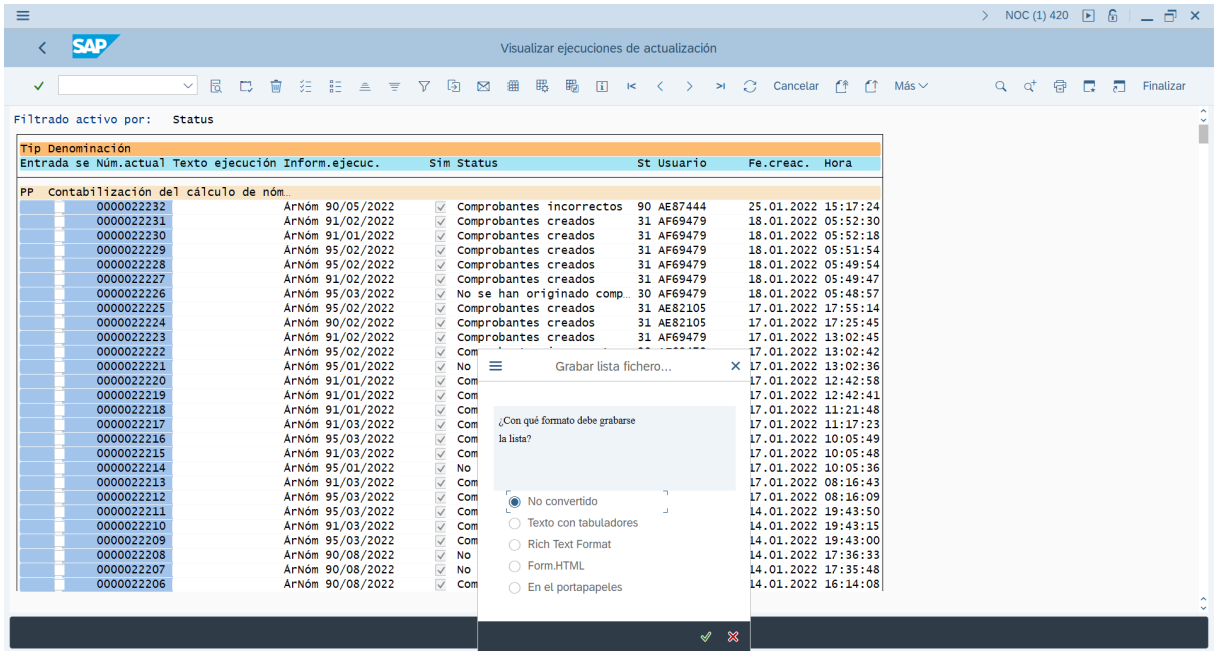


Figura 34: Os resultados da transação são exportados para um ficheiro

B.6 Consultar o uso do conceito de folha de pagamento

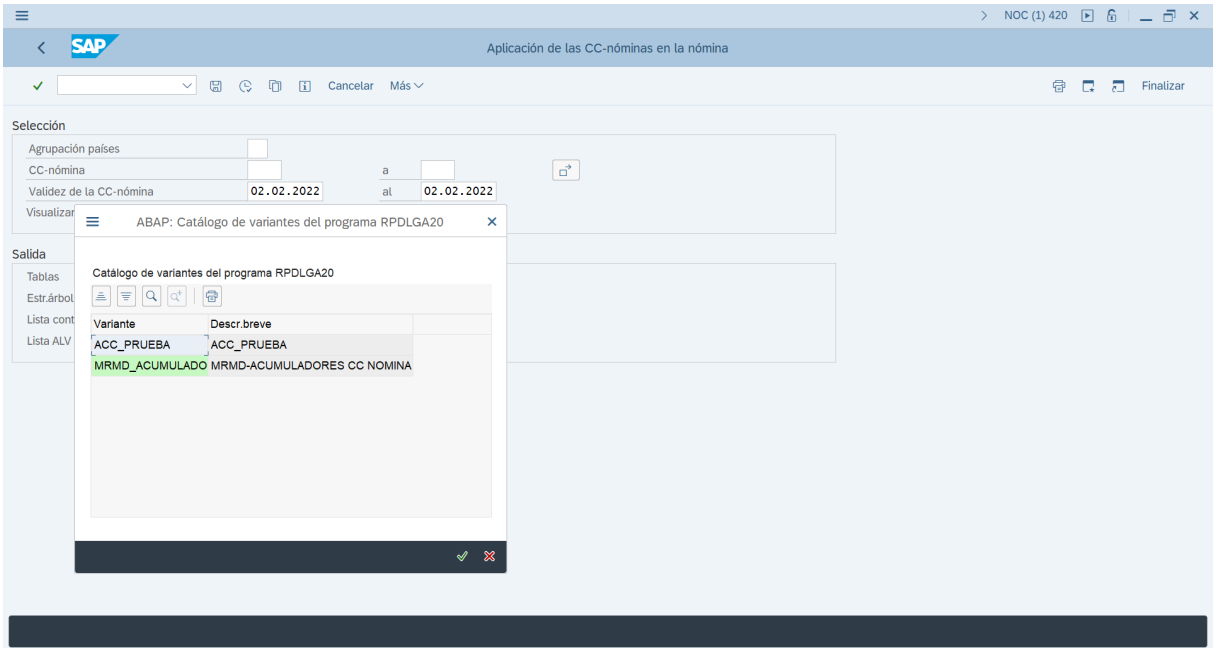


Figura 35: Escolha do variante para a transação pc00_m99_dlga20

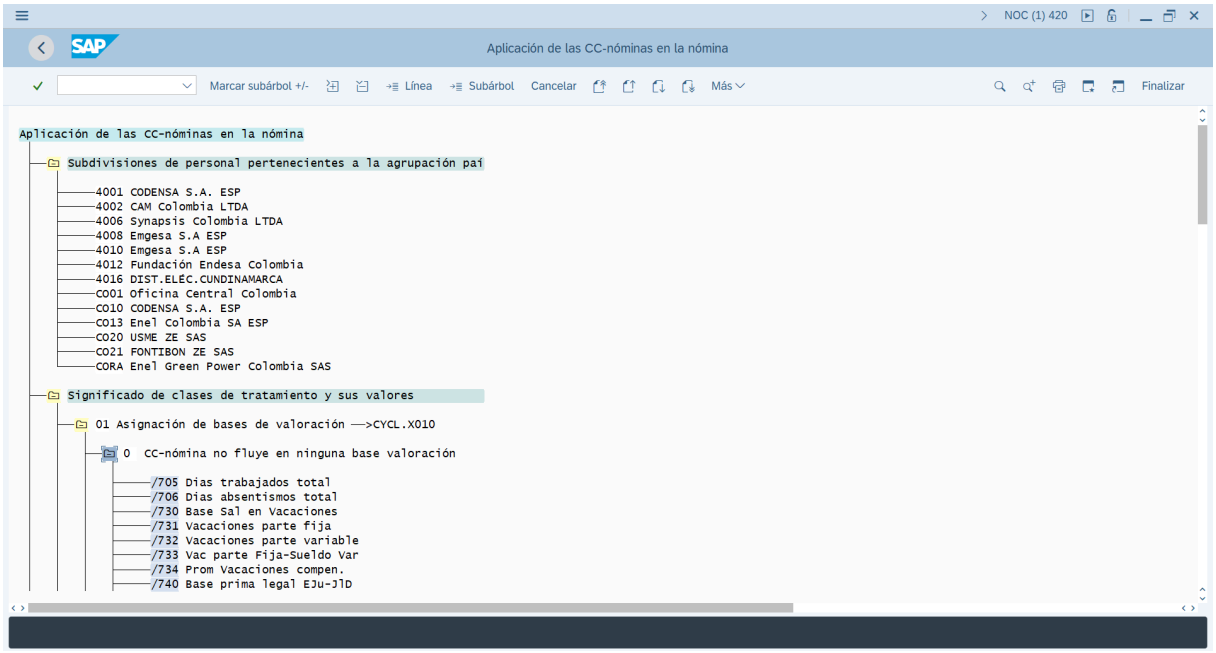


Figura 36: Os resultados da transação são exportados para um ficheiro

B.7 Verificar a ativação dos serviços de fluxo de trabalho

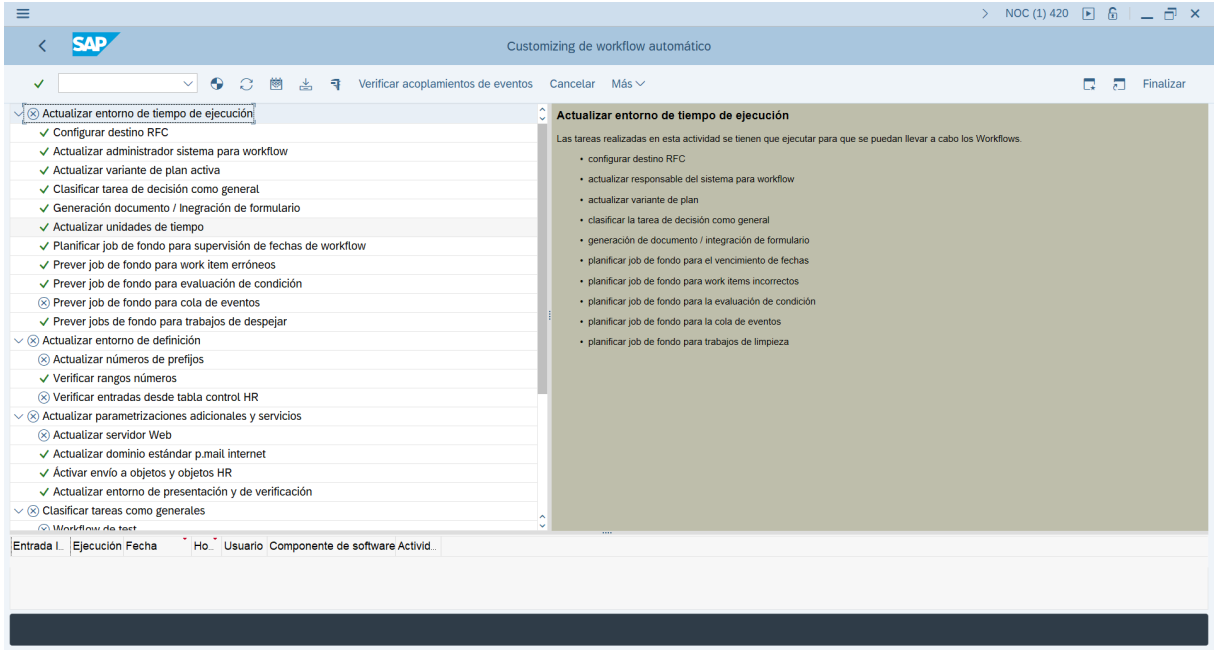


Figura 37: *Printscreen* do resultado da transação SWU3