UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

LINGUAGENS DE PROGRAMAÇÃO PARA NUVEM - EXPERIÊNCIAS COM GOOGLE APPS SCRIPT

TRABALHO DE GRADUAÇÃO

Júlio César Vieira

Santa Maria, RS, Brasil

2014

LINGUAGENS DE PROGRAMAÇÃO PARA NUVEM - EXPERIÊNCIAS COM GOOGLE APPS SCRIPT

Júlio César Vieira

Trabalho de Graduação apresentado ao Curso de Ciência da Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para a obtenção do grau de

Bacharel em Ciência da Computação

Orientadora: Profª. Drª. Andrea Schwertner Charão

Trabalho de Graduação N. 364
Santa Maria, RS, Brasil

2014

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

LINGUAGENS DE PROGRAMAÇÃO PARA NUVEM - EXPERIÊNCIAS COM GOOGLE APPS SCRIPT

elaborado por
Júlio César Vieira

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

Andrea Schwertner Charão, Drª.
(Presidente/Orientadora)

Benhur Stein, Prof. Dr. (UFSM)

Fernando Barbosa, M. Sc. (UFSM)

Santa Maria, 20 de Janeiro de 2014.

AGRADECIMENTOS

Aos meus pais, por terem me permitido estudar o curso que desejava e me dar a oportunidade de fazer isso em Santa Maria.

Aos meus amigos de Três Passos, alguns anteriores à faculdade e outros que surgiram durante, mas todos importantes para mim ao longo desses anos. Em especial à Vivian, à Lara, ao Caio e ao Jeiel.

Aos meus amigos de Santa Maria, mesmo os que não continuaram o Curso. Penso que fizemos Ciência da Computação só para que esse grupo pudesse se conhecer.

Ao Programa de Educação Tutorial de Ciência da Computação (PET-CC). Os anos participando das atividades deste grupo me auxiliaram a crescer como aluno e como pessoa.

À professora Andrea Charão, pelo trabalho como minha orientadora e por ter encontrado para mim um trabalho com o qual tive grande afinidade.

À Janice e ao Marcelo, por terem ajudado tanto nos documentos necessários para minha formatura.

Ao Curso de Ciência da Computação da UFSM e todo seu corpo discente e também aos demais professores de outros cursos que participaram da minha formação.

E a você, que sente que, de alguma forma, me auxilou durante estes quatro anos.

- Você acredita que pode mudar seu próprio destino? - perguntou ele, procurando uma resposta.
Alice se viu concordando com a cabeça.
- Senão, de que adianta? Se só estamos percorrendo um caminho preestabelecido, então todas as experiências que fazem de nós quem somos... amor, dor, alegria, aprender, dividir... de nada valeriam."
— O Labirinto - Kate Mosse

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

LINGUAGENS DE PROGRAMAÇÃO PARA NUVEM - EXPERIÊNCIAS COM GOOGLE APPS SCRIPT

AUTOR: JÚLIO CÉSAR VIEIRA
ORIENTADORA: ANDREA SCHWERTNER CHARÃO
Local da Defesa e Data: Santa Maria, 20 de Janeiro de 2014.

Atualmente, a expressão "Computação em Nuvem" já é largamente conhecida. A ideia de acessar dados em qualquer lugar sem depender exclusivamente de um dispositivo evolui a cada ano, com o aumento da disponibilidade de *softwares* e de espaço de armazenamento em Nuvem. A Google, que possui seu próprio sistema de Computação em Nuvem, o Google Apps, foi uma das pioneiras em outro serviço: Programação em Nuvem, considerado um novo paradigma de programação. Através do Google Apps Script, a Google tornou possível a implementação de programas na plataforma Google Apps na Nuvem. Desta forma, ao invés de executar o programa no lado do cliente, ele é executado na Nuvem da Google. A principal utilidade do Google Apps Script é realizar tarefas automaticamente entre os produtos da Google e até mesmo com serviços de terceiros. O objetivo principal deste trabalho é explorar o conceito dessa tecnologia de Programação em Nuvem. Com isso, buscou-se desenvolver aplicativos que possam ser utilizados em conjunto com serviços da Google usados pelos Cursos de Ciência da Computação e Sistemas de Informação da Universidade Federal de Santa Maria. Esses aplicativos desenvolvidos servirão para automatizar tarefas repetitivas que atualmente são executadas manualmente, consumindo tempo de funcionários e poderão ser utilizados a partir do primeiro semestre letivo de 2014.

Palavras-chave: Programação em Nuvem. Google Apps Script. Automatização.

ABSTRACT

Undergraduate Final Work
Undergraduate Program in Computer Science
Federal University of Santa Maria

PROGRAMMING LANGUAGES FOR THE CLOUD - EXPERIMENTS WITH GOOGLE APPS SCRIPT

AUTHOR: JÚLIO CÉSAR VIEIRA
ADVISOR: ANDREA SCHWERTNER CHARÃO
Defense Place and Date: Santa Maria, January 20st, 2014.

Nowadays, the term "Cloud Computing" is already widely know. The idea of access data anywhere without relying solely on one device evolves every year, with the increased availability of *softwares* and storage space in the Cloud. Google, which has its own Cloud Computing system, the Google Apps, was one of the pioneers in another service: Cloud Programming, considerated as a new programming paradigm. Through Google Apps Script, Google has made possible to implement programs in the Google Apps' Cloud platform. Thus, instead of executing the program in the client's side, it runs on the Google Cloud. The main use of Google Apps Script is to automatically accomplish tasks between Google's products and even with third-party services. The goal of this work is to explore the concept of this tecnology of Cloud Programming. Thus, it was aimed to develop applications that can be used with Google's services used by courses of Computer Science and Information Systems of the Federal University of Santa Maria. These application will be used to automate repetitive tasks thar are currently performed manually, consuming time of the employees and can be used from the first academic semester of 2014.

Keywords: Cloud Programming, Google Apps Script, Automation.

LISTA DE FIGURAS

2.1 Exemplo de código no editor do GAS. Fonte: (Google Apps Script, 2013a) . 17
3.1 Versão do script utilizando SpreadsheetApp 22
3.2 Versão do script utilizando DocumentApp 22
3.3 Tempos de início e fim de execução da primeira (a) e segunda (b) versões ... 23
3.4 Esquema de entrada e saída do primeiro caso 24
3.5 Interface do filtro utilizado no arquivo *inf.ldif* 25
3.6 Planilha com as URL's de todos os arquivos utilizados nos scripts 26
3.7 Resultado do pré-processamento no relatório dos alunos 27
3.8 Trecho do script que separa os alunos em turmas..... 28
3.9 Trecho do script que cria os grupos e insere os membros 30
3.10 Tempos de execução dos testes realizados 32
3.11 Planilha de referência para os semestres 33
3.12 Esquema de entrada e saída do segundo caso 34
3.13 Esquema de entrada e saída do terceiro caso 35
3.14 Função *criaAgenda* 36
3.15 Planilha com informações de datas necessárias para o script 37
4.1 Visualização dos grupos criados através do script 39
4.2 Membros do grupo Disciplina ELC1008..... 39
4.3 Tabela de horários do segundo semestre de Ciência da Computação..... 40
4.4 Agenda do segundo semestre de Ciência da Computação 41
4.5 Detalhes do evento quando ele é selecionado 41

LISTA DE TABELAS

2.1 Serviços disponíveis para integração no Google Apps Script. Fonte: (Google, 2013a) 16

LISTA DE APÊNDICES

APÊNDICE A – Código de Pré-Processamento do Arquivo *inf.ldif* 49
APÊNDICE B – Script Turmas 53
APÊNDICE C – Script Grupos 55
APÊNDICE D – Script Tabelas de Horários 56
APÊNDICE E – Script Agendas do Google Calendar 60

SUMÁRIO

1 INTRODUÇÃO 12
1.1 Objetivos 13
1.1.1 Objetivos Gerais 13
1.1.2 Objetivos Específicos 13
1.2 Justificativa 14
2 FUNDAMENTOS E REVISÃO TEÓRICA 15
2.1 Programação em Nuvem 15
2.2 Google Apps Script 15
3 DESENVOLVIMENTO 19
3.1 Casos Considerados 19
3.1.1 Criação automática de grupos 19
3.1.2 Geração de tabelas de horários 20
3.1.3 Criação de Agendas no Google Calendar 20
3.2 Experiências Preliminares 21
3.3 Criação automática de grupos 23
3.3.1 Descrição dos dados de entrada 23
3.3.1.1 Relatório de matriculados por disciplina 24
3.3.1.2 Índice dos nomes e *logins* 24
3.3.1.3 Relatório de disciplinas e professores 25
3.3.2 Implementação do Script 25
3.3.2.1 Script das Turmas 25
3.3.2.2 Script dos Grupos 29
3.4 Geração de Tabelas de Horários 32
3.4.1 Descrição dos dados de entrada 33
3.4.2 Implementação do Script 33
3.5 Criação de Agendas no Google Calendar 35
3.5.1 Implementação do Script 35
4 RESULTADOS E DISCUSSÃO 38
4.1 Resultados 38
4.1.1 Primeiro caso 38
4.1.2 Segundo caso 39
4.1.3 Terceiro caso 40
4.2 Discussão 41
4.2.1 Abrindo arquivos 41
4.2.2 Tempo de escrita 42
4.2.3 Limite de tempo de execução 42
4.2.4 Criando e inserindo dados em tabelas 42
4.2.5 Compartilhamento das agendas 43
5 CONCLUSÃO 44
REFERÊNCIAS 46
APÊNDICES 48

1 INTRODUÇÃO

Computação em Nuvem (*Cloud Computing*) é um modelo para permitir acesso ubíquo e conveniente sob demanda a recursos computacionais compartilhados (MELL; GRANCE, 2011). Esse modelo provê não somente acesso a dados, como também a serviços.

Com a evolução da Computação em Nuvem, muitos serviços estão migrando ou já estão totalmente baseados nela. O advento de grandes serviços de *Cloud Storage*, tais como Google Docs (atualmente Google Drive); DropBox; iCloud, da Apple; e SkyDrive, da Microsoft, fornece o acesso a arquivos a partir de dispositivos como computadores pessoais, smartphones e notebooks em qualquer lugar com conexão à Internet disponível. Assim, pessoas conseguem acessar seus dados e arquivos necessários para trabalho ou diversão sem depender exclusivamente de seu computador pessoal em casa ou no trabalho.

Um dos modelos atuais de serviços presentes na Computação em Nuvem é chamado de *Platform as a Service* (PaaS – Plataforma como um Serviço, em português). Esse serviço provê a capacidade do usuário de desenvolver aplicativos dentro da infraestrutura da Nuvem utilizando linguagens de programação, bibliotecas, serviços e ferramentas criadas pelo fornecedor (MELL; GRANCE, 2011). Neste modelo, o consumidor não gerencia ou controla qualquer infraestrutura da Nuvem: ele controla apenas o desenvolvimento de aplicações e possíveis configurações de ambiente (MELL; GRANCE, 2011).

Seguindo este modelo, surgiu a Programação em Nuvem (*Cloud Programming*), uma área em crescimento nos últimos anos, que possibilita o desenvolvimento de programas em *Integrated Development Environments* (IDE - Ambientes de Desenvolvimento Integrado, em português) baseados em linguagens para Nuvem. A principal ideia dessa nova tecnologia é fornecer um ambiente para programação sem depender de um dispositivo específico, tornando o programador capaz de acessar seu trabalho tanto no serviço, como em casa e durante viagens.

Como exemplos de serviços que suportam Programação em Nuvem, pode-se citar o Google Apps (Google Developers, 2013) e o Windows Azure (Microsoft, 2013). Lançado em 2006, o Google Apps é um serviço da Google que provê versões independentes e customizáveis de diversos produtos Google e está disponível em versões para empresas (Google Apps for Business) e para instituições de ensino (Google Apps for Education). Já o Windows Azure, lançado em 2010, foi criado para construir, desenvolver e gerenciar aplicações e serviços e suporta o desenvolvimento em várias linguagens, tais como .NET, PHP, Java e Python.

Um destes serviços do Google Apps é o Google Apps Script (GAS, nome utilizado para descrever tanto o serviço, quanto a linguagem que o mesmo oferece). Criado e mantido (está em constante atualização) pela Google, o GAS busca integrar os serviços do Google Apps através de scripts desenvolvidos em JavaScript. Apesar de relativamente nova (lançada em 2009), essa linguagem apresenta várias ferramentas para programação, contando com serviços do Google Apps, como Calendar, Gmail, Contacts, Domain, Drive, Groups, entre outros. O principal objetivo desta linguagem de *Cloud Programming* é automatizar tarefas que integrem um ou mais serviços da Google (Google Support, 2013).

Na Universidade Federal de Santa Maria, o Curso de Bacharelado em Ciência da Computação utiliza, desde 2007, o serviço Google Apps associado ao domínio *inf.ufsm.br* para serviços de e-mail, fóruns de discussão, entre outros (SCHEID et al., 2012). O Curso de Sistemas de Informação, desde a sua criação em 2009, compartilha o mesmo domínio. Com o aumento de recursos e informações, como tabelas de dados, utilizados pelas Coordenações e Secretarias dos Cursos, a realização de tarefas tende a se tornar repetitiva, consumindo de funcionários tempo que poderia ser melhor utilizado caso essas tarefas fossem automatizadas. Através do Google Apps Script, é possível automatizar algumas dessas tarefas, a fim de otimizar o trabalho realizado.

1.1 Objetivos

1.1.1 Objetivos Gerais

O objetivo geral deste trabalho é explorar o conceito de Programação em Nuvem, o qual é recente e apresenta-se como uma tendência para os próximos anos. Outro objetivo geral é realizar estudos de caso com o Google Apps Script, buscando aplicar esta tecnologia a tarefas realizadas na plataforma Google Apps, associada ao domínio *inf.ufsm.br*. Com isso, busca-se adquirir conhecimento sobre as vantagens e limitações dos recursos de Programação em Nuvem oferecidos por essa plataforma.

1.1.2 Objetivos Específicos

Como objetivo específico, tem-se a intenção de desenvolver e disponibilizar três aplicações para uso tanto da instituição a qual o aluno é vinculado, quanto para outras que desejarem. A ideia principal dessas aplicações é que elas possam vir a ser usadas no âmbito do domínio

inf.ufsm.br.

1.2 Justificativa

As aplicações que desenvolvidas visam a automatização de tarefas realizadas pelas Coordenações dos Cursos de Ciência da Computação e de Sistemas de Informação. Esse tipo de aplicação foi escolhido por dois principais motivos. Primeiramente, o domínio *inf.ufsm.br* pode ser considerado grande (aproximadamente 350 usuários) e estável (não é propenso a erros de funcionamento). O segundo motivo, justamente em decorrência do primeiro, é que o domínio se caracteriza como um ambiente ideal para testes com o GAS, já que sem um domínio não seriam possíveis experiências utilizando algumas das funcionalidades disponíveis.

2 FUNDAMENTOS E REVISÃO TEÓRICA

Este capítulo abordará, primeiramente, o conceito de Programação em Nuvem. Após, será feita uma revisão da linguagem de programação utilizada neste trabalho, a Google Apps Script.

2.1 Programação em Nuvem

A ideia básica da Computação em Nuvem consiste em deslocar os servidores de instituições para terceiros gerenciarem o armazenamento de dados, retirando da própria instituição o gerenciamento de escalabilidade, tolerância a falhas e demais preocupações que envolvem grandes servidores.

A partir do momento em que os serviços de uma empresa estão na Nuvem, surgiu a possibilidade de transportar também o desenvolvimento destes serviços para a ela. Deste modo, nem mesmo os computadores utilizados para desenvolver aplicativos precisariam ter grande poder computacional, já que toda a necessidade de processamento para compilação e execução seria provida pelas empresas que armazenariam estes softwares. Assim surgiu a Programação em Nuvem.

Considerada uma das 10 tecnologias emergentes mais importantes, a Programação em Nuvem traz a ideia de converter programas existentes para aplicativos na Nuvem (Naone, Erica, 2010). O Microsoft Windows Azure, por exemplo, já é utilizado por grandes empresas, tais como a Coca-Cola e o eBay, para hospedar seus sites e serviços (Chou, David, 2009).

2.2 Google Apps Script

A Google Apps Script é uma linguagem de Programação em Nuvem baseada em JavaScript e permite utilizar o Google Apps construindo aplicações *Web* (Google Support, 2013). Para isso, são desenvolvidos scripts em um editor baseado em navegadores *Web* e que são armazenados e executados nos servidores do Google (Google, 2013a). Esta é uma das inovações do Google Apps Script: o desenvolvedor não precisa de nada além de um navegador para escrever e executar seus scripts.

Isto implica em uma mudança de paradigma de programação: abandonar grandes pacotes de software de compilação, utilizados, por exemplo, para Java e C++, e adotar um simples

navegador *Web* para utilizar como editor, compilador e executor. O desenvolvedor não mais precisa se preocupar em adquirir e configurar um IDE, pois todas as ferramentas necessárias para o desenvolvimento de aplicações são fornecidas.

Lançada em 2009, o GAS ainda está em constante desenvolvimento, com novos recursos sendo implementados frequentemente. Na Tabela 2.1, é possível visualizar os serviços do Google Apps que podem ser usados pelo GAS para gerar scripts que realizem automaticamente uma tarefa, definida no script. Para utilizar a linguagem e criar scripts, é necessário possuir uma conta Google ou Google Apps, além de um navegador *Web* suportado (Google Chrome, Mozilla Firefox, Safari ou Internet Explorer).

Tabela 2.1: Serviços disponíveis para integração no Google Apps Script. Fonte: (Google, 2013a)

Serviços Google Apps	**Serviços Google Avançados**	**Serviços de Script**
Calendar	Admin SDK	Base
Contacts	AdSense	Cache
DocsList	Analytics	Charts
Document	BigQuery	Content
Domain	Fusion Tables	HTML
Drive	Google+ Domains	JDBC
Finance	Mirror	Lock
Forms	Prediction	Mail
Gmail	Tasks	Properties
Groups	URL Shortener	Script
Language	YouTube	ScriptDB
Maps	YouTube Analytics	UI
Sites		URL Fetch
Spreadsheet		Utilities
		XML

Como dito anteriormente, a linguagem GAS é baseada em JavaScript, contendo todas as funções básicas, acrescidas as funcionalidades específicas da linguagem GAS. Já que JavaScript é uma linguagem similar à Java, programadores com experiência em Java encontram muita facilidade para iniciar a programação no GAS. Programadores que nunca tiveram contato com JavaScript e tiveram pouco ou nenhum contato com Java também não deverão ter muitas dificuldades para começar a programar, visto que JavaScript é uma linguagem relativamente simples, orientada a objetos e de tipagem fraca, ou seja, não é necessário indicar o tipo de dado das variáveis.

Um script, resumidamente, nada mais é que uma série de instruções para realizar uma tarefa em particular. Esse script poderá ser executado sob circunstâncias pré-determinadas,

como, por exemplo: toda segunda-feira, todo dia primeiro do mês, a cada N minutos ou manualmente, quando o usuário desejar. Ao acessar a página inicial do Google Apps Script, faz-se o *login* com a conta Google e o usuário é direcionado imediatamente ao ambiente do editor de scripts. O editor possui *template* prontos para determinadas funções, dependendo da intenção do programador. Um exemplo de *templates* pode ser visto na Figura 2.1, o qual serve para tarefas que utilizem arquivos dentro do Google Drive do usuário, ferramenta da Google para armazenamento na Nuvem. Todos os scripts que o usuário desenvolver ficarão armazenados no seu Google Drive.

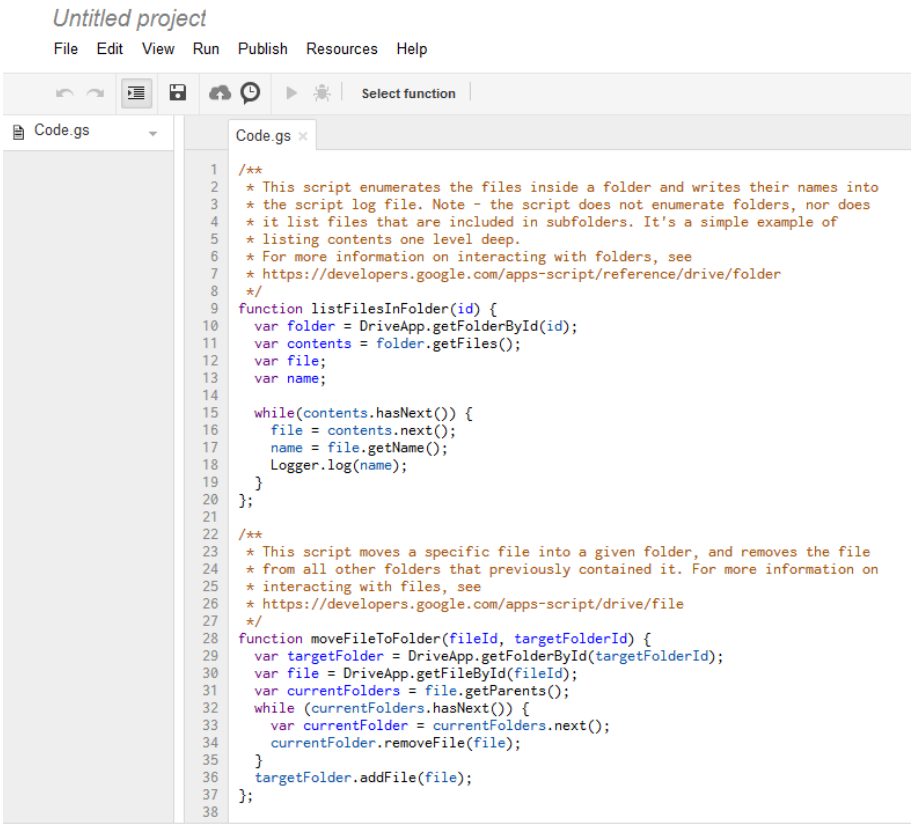


Figura 2.1: Exemplo de código no editor do GAS. Fonte: (Google Apps Script, 2013a)

Na Figura 2.1 também é possível visualizar outros detalhes relativos ao editor de scripts do GAS. No campo que está escrito *"Untitled project"*, sendo este o nome padrão designado no momento de criação, pode-se alterar o nome do projeto. Nos menus, é possível acessar opções de tratamento do arquivo, de edição do código, de execução, entre outras. Os ícones rápidos servem como atalhos para opções mais utilizadas, tais como *Save*, *Run* e *Debug*.

Uma funcionalidade interessante que terá papel importante no desenvolvimento são as *Project Properties* (Propriedades do Projeto), acessadas a partir do menu *File*. Nestas propriedades, além de informações como nome e descrição do projeto, também encontram-se as *User*

Properties e *Project Properties* (Propriedades do usuário e do projeto, respectivamente). Nelas é possível salvar valores (como em variáveis) que são mantidos mesmo após o script terminar de executar. A diferença entre elas consiste em propriedades do projeto terem permissão de acesso somente do projeto que as criou. Já as propriedades do usuário podem ser acessadas por todos os projetos do usuário. As propriedades do projeto são extremamente úteis em casos de que se precisa passar um dado entre execuções do mesmo script, enquanto que as do usuário são utilizadas para passar dados entre projetos diferentes.

O serviço Google Apps Script impõe algumas limitações de uso para evitar o abuso por parte de usuários (Google, 2013b). As limitações atingem tarefas como atividades com e-mails (anexar arquivos, tamanho de mensagens e envios), com banco de dados e com outras APIs do Google. Além disso, também há um limite de tempo de execução, que, como será mostrado no Capítulo 3, causou problemas e incentivou a busca de soluções com processamento mais rápido.

3 DESENVOLVIMENTO

Este capítulo abordará as etapas de desenvolvimento realizadas buscando atingir os objetivos do trabalho. Na primeira etapa foram decididas quais seriam as aplicações a serem desenvolvidas durante o trabalho. A segunda etapa consistiu na aquisição de conhecimento em relação ao serviço e à linguagem Google Apps Script, através da realização de experiências preliminares com algumas das ferramentas que seriam utilizadas futuramente. Nas etapas seguintes, iniciou-se o desenvolvimento dos casos considerados.

Como os Cursos de Ciência de Computação e Sistemas de Informação compartilham o mesmo domínio *inf.ufsm.br* e também algumas das suas disciplinas (por exemplo, uma disciplina ofertada por Ciência da Computação possui alunos de Sistemas de Informação, e vice-versa), foram utilizados disciplinas e alunos de ambos os Cursos.

3.1 Casos Considerados

Como foi dito nos objetivos específicos, a ideia é que sejam desenvolvidas três aplicações, descritas a seguir.

3.1.1 Criação automática de grupos

Os alunos de ambos os Cursos, ao se matricularem, recebem uma conta de e-mail no domínio *inf.ufsm.br*, fornecida pela Coordenação. O principal objetivo deste e-mail é fornecer um meio de comunicação acadêmico entre os alunos e entre alunos e professores. No entanto, quando a intenção é comunicar-se com todos os matriculados em certa disciplina, a tarefa torna-se complicada, tanto para professores, que precisam encontrar todos os e-mails através de uma lista de matriculados, quantos para os próprios alunos, que não possuem um método simples para descobrirem todos os seus colegas.

Para contornar este problema, a criação de listas de e-mails (chamadas de grupos) pode ser usada. Desta forma, quando alguém (aluno ou professor) quiser se comunicar com uma turma inteira, somente precisará enviar um e-mail para um grupo e este encaminhará a mensagem para todos os seus memgros.

A única ferramenta pronta que é disponibilizada no momento pela Google para criar grupos é manual, através de um browser, onde o usuário tem que criar e adicionar manualmente

os e-mails na lista. E como são várias disciplinas todo semestre, este trabalho tomaria muito tempo do responsável que criaria os grupos. O GAS tem a vantagem de, além de poder realizar automaticamente a tarefa, não há necessidade de autenticação de um serviço externo, pois, para ser executado um script, é necessário estar conectado a sua conta Google.

Ao automatizar essa tarefa, além do ganho em tempo útil do funcionário incumbido dela, ela torna-se à prova de erros humanos, como errar um *login*, inserir um aluno no grupo errado ou esquecer de um aluno em um grupo.

3.1.2 Geração de tabelas de horários

Nos Cursos de Ciência da Computação e Sistemas de Informação, as Coordenações buscam, sempre que possível, atender demandas de alunos para horários de disciplinas. Assim, os horários não são fixos, variando a cada semestre. O aluno, após fazer a matrícula, pode visualizar no Portal do Aluno da UFSM seu quadro de horários do semestre. Apesar do nome, os horários são mostrados como uma planilha. Essa planilha é fornecida pela Sistema de Informações Educacionais (SIE), porém, os dados que estão no SIE, como horários de aulas e professores, são inseridos manualmente por funcionários dos Cursos.

As Coordenações, para auxiliar os alunos, especialmente os calouros, disponibilizam verdadeiros quadros de horários, divididos por semestre, mostrando os horários das disciplinas durante a semana, seu professor e sua sala. Esses quadros ficam disponíveis no mural dos Cursos e no site *inf.ufsm.br*. Todo o serviço empenhado para estruturação dos quadros é manual, o que, novamente, demanda tempo e está propenso a erros. Através do Google Apps Script, encontrou-se uma solução automatizada de confeccção destes quadros para, desta forma, facilitar outra tarefa repetitiva realizada dentro do domínio *inf.ufsm.br*. O quadro de horários gerado poderá, da mesma forma, ser exposto nos murais e também no site citado.

3.1.3 Criação de Agendas no Google Calendar

Este caso foi considerado, primeiramente, por utilizar um serviço Google diferente dos serviços dos outros casos: o Google Calendar (Google Agenda, em português). Os outros casos baseiam-se, principalmente, em operações com arquivos de texto e planilha e com grupos de um domínio.

A ideia deste caso é similar ao caso exposto no item anterior, mas, ao invés de gerar tabelas de horários em arquivos texto, pretende-se criar eventos para as aulas, em agendas

dos dois cursos separadas por semestre. Deste modo, o aluno poderia verificar seus horários através da conta Google que lhe é dada desde o início do Curso. Outra possibilidade seria o aluno criar notificações para esses eventos e, dessa forma, poder ser avisado, por exemplo, pelo *smartphone* dos horários de suas próximas aulas. O evento no Google Calendar também poderia ser usado para adicionar anotações para as próximas aulas, como ”Entrega de trabalho”, ”Prova” ou ”Aula de dúvidas”. A possibilidade de ter uma agenda eletrônica com todos os seus horários facilmente acessíveis também auxiliaria na organização pessoal do aluno.

3.2 Experiências Preliminares

Uma vez que a tecnologia Google Apps Script era totalmente desconhecida pelo aluno, fez-se necessário um período de aquisição de conhecimento e realização de experiências para colocar o aluno a par das ferramentas que seriam utilizadas no decorrer do projeto.

A atividade proposta foi realizar um levantamento de todos os grupos dentro do domínio *inf.ufsm.br* e seus respectivos usuários. Essa atividade foi indicada por realizar operações com classes que seriam usadas posteriormente, como:

- *GroupsManager*: classe de operações com grupos dentro de um domínio;
- *SpreadsheetApp*: classe de operações com planilhas;
- *DocumentApp*: classe de operações com documentos de texto;

Foram desenvolvidas duas versões. Na primeira, como pode ser vista na Figura 3.1, foram utilizadas as classes *GroupsManager* e *SpreadsheetApp*. A função deste script é, resumidamente, coletar o nome e os *logins* de todos os grupos e seus membros dentro do domínio *inf.ufsm.br*. O arquivo resultante da execução teste script contém *n* abas (sendo *n* o número de grupos no domínio) e cada uma delas possui o *login* de um membro do grupo por célula. A função *Logger.log* é utilizada para escrever dados na forma de texto no *log* do script e sua utilização será explicada em breve.

Inicialmente desenvolvida apenas para fins de conhecimento, a versão utilizando *DocumentApp*, vista na Figura 3.2, mostrou-se mais útil do que o esperado. Ao terminar sua implementação, foi notado uma maior velocidade de execução desta versão em relação à outra.

Como ambas as versões possuem implementação praticamente idêntica, tendo suas diferenças somente quanto ao tipo de arquivo utilizado para escrever a saída, ficou claro que

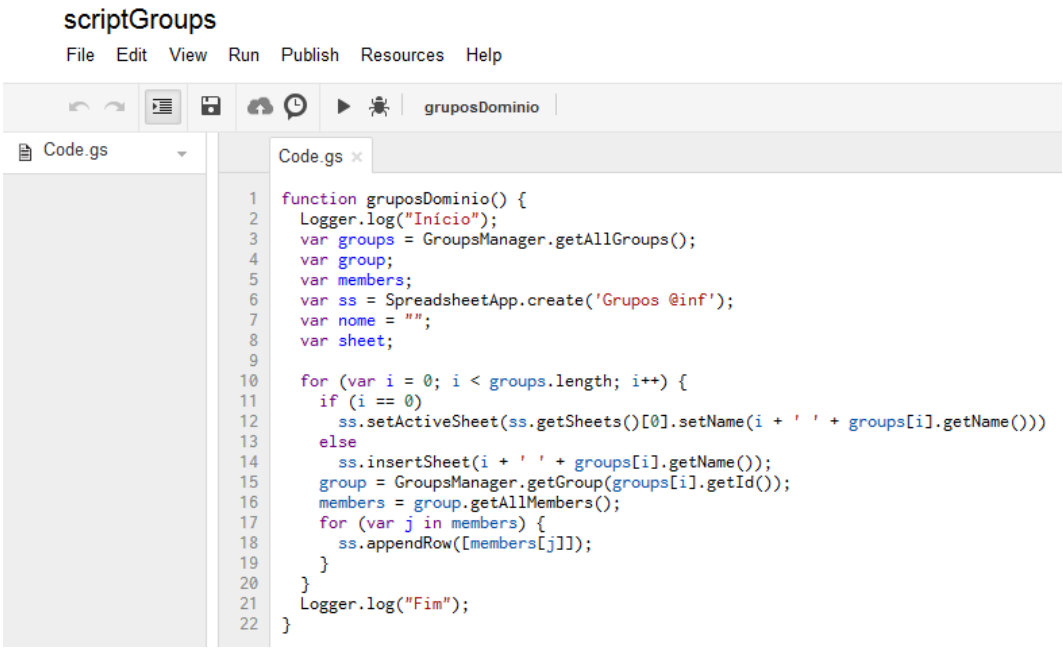


Figura 3.1: Versão do script utilizando SpreadsheetApp

escrever dados em planilhas é mais custoso em termos de tempo de execução do que escrever em documentos de texto.

Essa diferença de tempo de execução ficou evidenciada quando, após acrescentar em

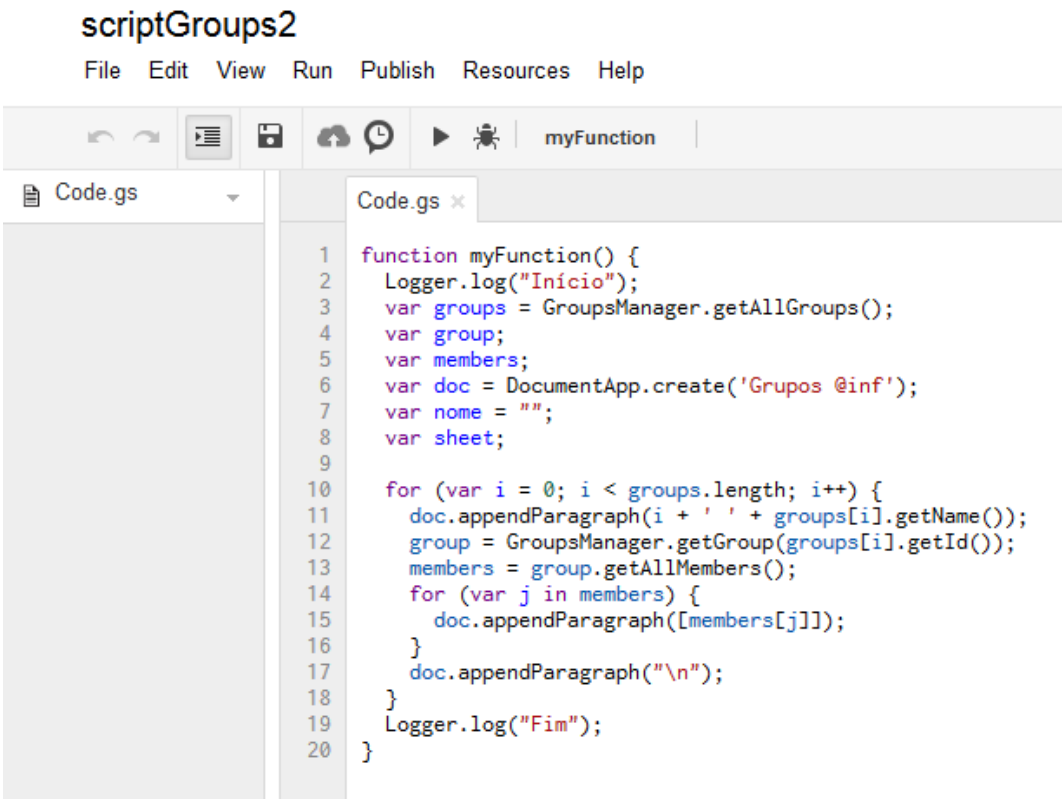


Figura 3.2: Versão do script utilizando DocumentApp

ambas as versões as chamadas de função *Logger.log(data)* para ”Início” e ”Fim”, o tempo que a primeira versão (com *SpreadSheetApp*) levou quase o dobro de tempo que a segunda. Esse teste foi repetido dez vezes para evitar que os dados fossem prejudicados por possíveis variações no tempo de execução causadas por picos de processamento do serviço GAS. Os resultados se mantiveram relativamente próximos ao mostrado na Figura 3.3.

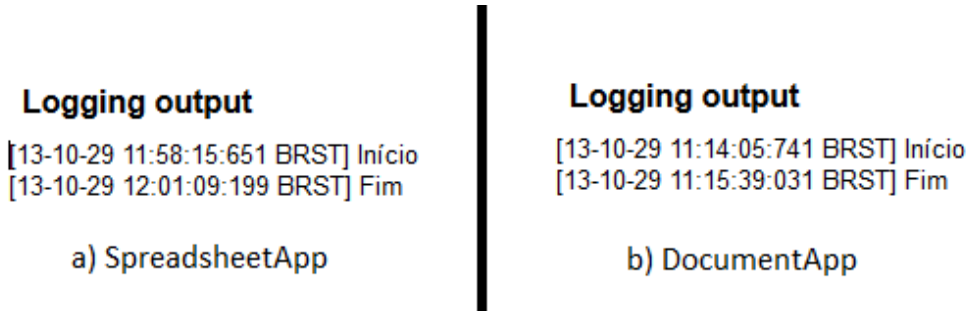


Figura 3.3: Tempos de início e fim de execução da primeira (a) e segunda (b) versões

A descoberta de que há diferença no tempo de escrita entre os dois tipos de arquivo resultou em uma mudança na posterior etapa de desenvolvimento.

3.3 Criação automática de grupos

A primeira parte do trabalho foi o desenvolvimento de um script que criasse, a partir de dados de entrada, listas de e-mail no domínio *inf.ufsm.br* para cada turma, a fim de facilitar a comunicação entre os alunos e os professores das turmas. Os passos seguidos estão explicados nos itens a seguir.

3.3.1 Descrição dos dados de entrada

Antes de começar o desenvolvimento do script, precisava-se obter os dados necessários, tais como códigos de disciplinas, *logins* de alunos e de professores, quais os alunos estão matriculados em determinada disciplina e o seu respectivo professor. Nas subseções a seguir serão descritos os arquivos utilizados para obter estes dados e qual o pré-processamento utilizado, caso necessário. Todos os arquivos foram obtidos através da Coordenação do Curso de Ciência da Computação da UFSM. A Figura 3.4 mostra uma esquematização de como funcionará o script: quais são os dados de entrada e qual é a saída.

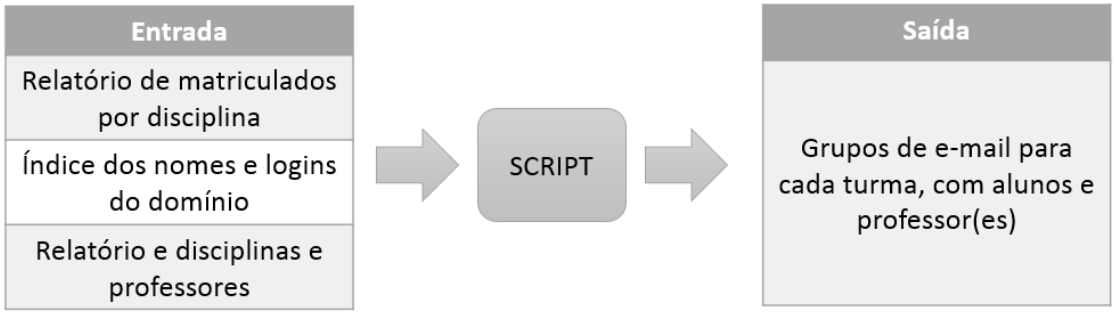


Figura 3.4: Esquema de entrada e saída do primeiro caso

3.3.1.1 Relatório de matriculados por disciplina

Gerado a partir do banco de dados da Universidade através do SIE, este arquivo no formato de planilha possui trinta e dois (32) atributos, sendo que apenas dois destes são necessários para a tarefa proposta: o nome do aluno e o código da disciplina que ele está matriculado. Apesar de que um pré-processamento poderia ser realizado através de programas editores de planilhas, tais como Microsoft Excel, a eliminação de dados desnecessários desta planilha provou-se mais simples durante o início da execução do script.

3.3.1.2 Índice dos nomes e *logins*

Este arquivo contém as informações de todos os usuários do domínio *inf.ufsm.br*, tais como nome, *login*, *homeDirectory* para pasta pessoal de arquivos dentro do servidor do Curso, além de diversos outros atributos. Ele é gerado a partir de qualquer máquina dentro do domínio através do comando *shell script: ldapsearch -x -L -h ldap.inf.ufsm.br -b "dc=inf,dc=ufsm,dc=br" "(objectclass=*)"*. De todas as informações contidas no arquivo, tambem só são necessárias duas: o nome e o *login*. Por ser um arquivo grande (547.605 caracteres em 17.107 linhas), o Google Drive não suporta a criação de um arquivo deste tamanho e, portanto, foi necessário desenvolver um filtro externo ao Google Drive que diminuísse o tamanho do arquivo.

Por familiaridade com a linguagem, escolheu-se implementar o filtro em Java. O código da implementação deste filtro encontra-se disponível no Apêndice A e a interface pode ser vista na Figura 3.5. O filtro lê o arquivo *inf.ldif* e extrai dele apenas o nome do usuário que está no atributo *cn* e o *login* do usuário. Por motivos de maior simplicidade de implementação do script, o *login* é retirado do final do *homeDirectory* ao invés do atributo *uid*. Desta forma, o arquivo resultante contém, nesta ordem, o nome do aluno e seu respectivo *login*, necessário

posteriormente para adicionar seu e-mail ao grupo.

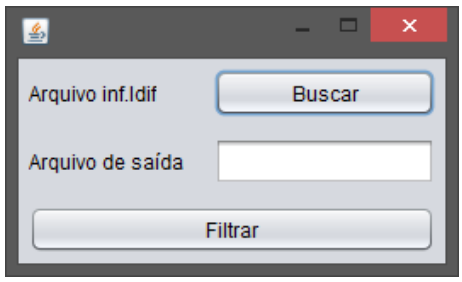


Figura 3.5: Interface do filtro utilizado no arquivo *inf.ldif*

3.3.1.3 Relatório de disciplinas e professores

Este arquivo, que também é gerado através do SIE, possui informações sobre as disciplinas ofertadas pelos dois Cursos, tais como código e nome da disciplina, vagas oferecidas, horários de aulas, etc., totalizando vinte e um (21) atributos. Destes, também são necessários apenas dois: o código da disciplina e o nome do professor (ou professores, caso haja mais de um) que a ministra. Da mesma forma que o relatório dos matriculados por disciplina, o pré-processamento é realizado na fase inicial do script.

3.3.2 Implementação do Script

Em posse de todos os dados necessários, iniciou-se a implementação do script que criará os grupos dentro do domínio. Apesar de não estar no planejamento inicial, devido ao tempo de execução, a tarefa foi separada em dois scripts: a primeira, utilizando os três arquivos descritos na seção 3.3.1, tem como saída documentos de texto contendo o *login* dos alunos matriculados e do professor. O nome destes documentos é o código da disciplina. A segunda parte lê o nome e o conteúdo destes arquivos e faz a criação do grupo e inserção dos alunos e do professor. O desenvolvimento e funcionamento de ambos são descritos a seguir e o código pode ser visto por completo nos Apêndices B e C.

3.3.2.1 Script das Turmas

Na atividade realizada como experiência utilizando GAS, foram usados os tipos *DocumentApp* e *SpreadsheetApp*, ambos para fins de escrita em arquivo. Porém, logo no início do desenvolvimento do primeiro script, foi necessário realizar leituras, tanto em documentos de texto (para o *inf.ldif*), quanto em planilhas (para os dois relatórios). Mas, primeiramente, é

preciso abrir os arquivos para leitura. Para se realizar isso, há dois modos:

- *openByUrl*: função que recebe como argumento o URL (*Uniform Resource Locator*, Localizador Padrão de Recursos) do arquivo, localizado no Google Drive do programador;
- *openById*: função que recebe como argumento um ID (identificador). Este ID faz parte do URL do arquivo.

O método escolhido foi o primeiro. Para facilitar a utilização futura, escolheu-se por colocar todos as URL's que serão utilizadas em todos os scripts em uma planilha (Figura 3.6). O script, ao necessitar de algum arquivo, leem essa planilha e selecionam o URL do arquivo necessário. Essa estratégia visa facilitar a execução dos scripts por pessoas leigas em programação, evitando, assim, que sejam necessárias alterações diretamente no código. Após os arquivos serem abertos, as seguintes funções foram utilizadas para remover linhas e colunas desnecessárias:

inf.ldif	docs.google.com/a/inf.ufsm.br/document/d/1Cy5jGlK0XVHqoi6XeifdLIeDOobVa5ziWOnkKeqGlVw/edit
Relatório dos matriculados	docs.google.com/a/inf.ufsm.br/spreadsheet/ccc?key=0AsgLylmbOlmLdFUwc2xUQi01UjFFb2VCVVEtRms4LUE&usp=drive_web#gid=0
Relatório dos professores (grupos)	docs.google.com/a/inf.ufsm.br/spreadsheet/ccc?key=0AsgLylmbOlmLdGVrX2JlcXkwRzJiZ0FYZ2g4MmtVZnc&usp=drive_web#gid=0
Relatório dos professores (horários)	docs.google.com/a/inf.ufsm.br/spreadsheet/ccc?key=0AsgLylmbOlmLdGtNZHJUTzVLZ0lJZ0sxcjF0VW96OHc&usp=drive_web#gid=0
Referência dos semestres	docs.google.com/a/inf.ufsm.br/spreadsheet/ccc?key=0AsgLylmbOlmLdGhMUnh6SnFKZlAxNmxCbHVrNl8yNGc&usp=drive_web#gid=0
Horários CC	docs.google.com/a/inf.ufsm.br/document/d/1GY2B6IzRlOYr9bHHdloovqQrnAN4QCO266LpwEDjRwo/edit
Horários SI	docs.google.com/a/inf.ufsm.br/document/d/1BEZx58E3dFXYlZC0lmlypK8WT0SjZUK9tGba1FJThyw/edit

Figura 3.6: Planilha com as URL's de todos os arquivos utilizados nos scripts

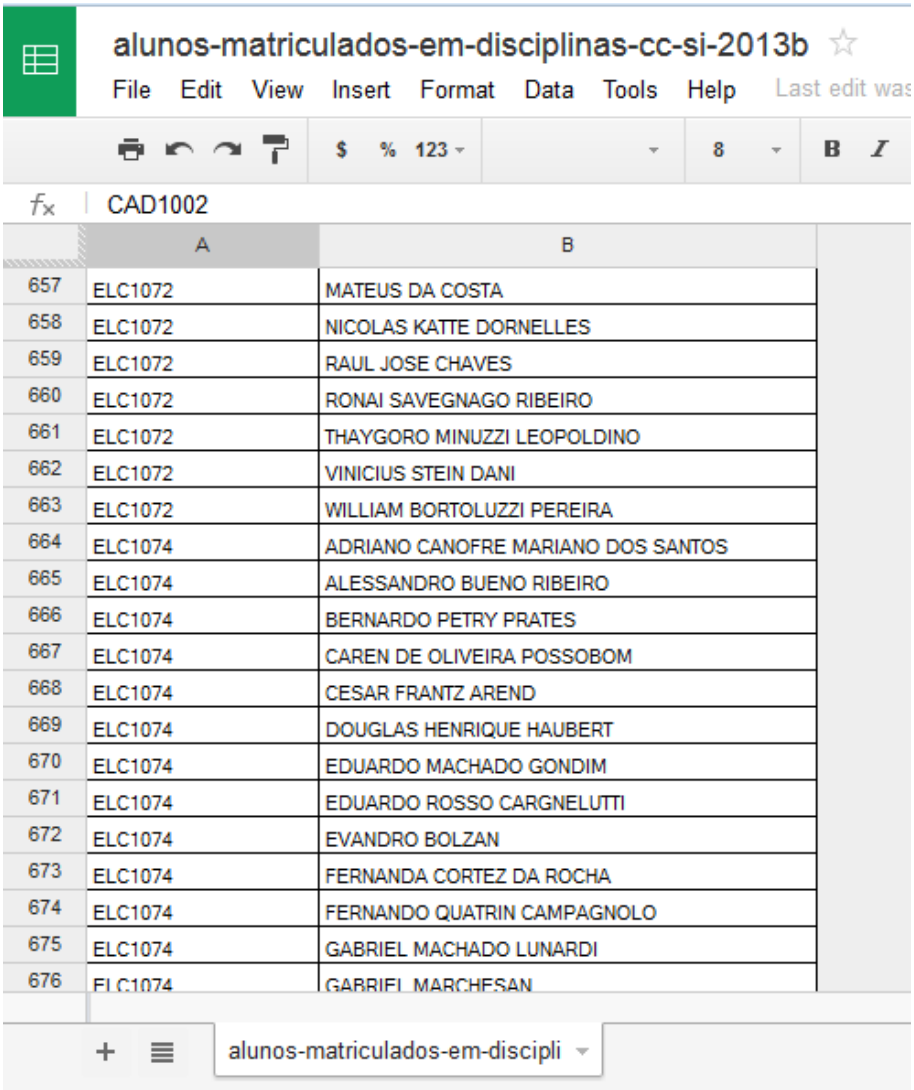
- *deleteColumn* e *deleteColumns*: ambas as funções servem para deletar uma ou mais colunas da planilha. Seus parâmetros são a coluna que se deseja eliminar ou a coluna que se deseja iniciar a eliminação e o total de colunas que será eliminado;
- *deleteRow*: deleta a linha especificada pelo parâmetro. No caso, como a primeira linha é o cabeçalho e sabe-se que dados cada coluna contém, não se faz necessário mantê-lo.

Para poder manipular os dados dentro das células da planilha, precisou-se primeiramente selecionar todas as células que possuem dados, utilizando os métodos:

- *getSheets*: retorna um vetor de *sheets*. Da forma que foi usada, seleciona especificamente o número do *sheet* informado. Para futuras referências, a palavra *sheet* será denominada por "aba" e a palavra *spreadsheet* será denominada por "planilha";
- *getDataRange*: essa função retorna um *Range*, que é um objeto que permite acessar e modificar os dados de planilhas de forma similar a uma matriz. A função seleciona um *Range* com todas as células que contém dados, o que, neste caso, é toda a planilha;

- *sort*: ordena alfabeticamente ou em ordem numérica crescente os dados da coluna *n*, especificada no parâmetro. Em ambos os arquivos, os dados são ordenados em relação ao código da disciplina. Isso torna o relatório dos alunos mais simples, colocando todos os matriculados na mesma turma um abaixo do outro.

Na Figura 3.7 é possivel ver o resultado da eliminação de colunas e da ordenação dos dados.



	A	B
657	ELC1072	MATEUS DA COSTA
658	ELC1072	NICOLAS KATTE DORNELLES
659	ELC1072	RAUL JOSE CHAVES
660	ELC1072	RONAI SAVEGNAGO RIBEIRO
661	ELC1072	THAYGORO MINUZZI LEOPOLDINO
662	ELC1072	VINICIUS STEIN DANI
663	ELC1072	WILLIAM BORTOLUZZI PEREIRA
664	ELC1074	ADRIANO CANOFRE MARIANO DOS SANTOS
665	ELC1074	ALESSANDRO BUENO RIBEIRO
666	ELC1074	BERNARDO PETRY PRATES
667	ELC1074	CAREN DE OLIVEIRA POSSOBOM
668	ELC1074	CESAR FRANTZ AREND
669	ELC1074	DOUGLAS HENRIQUE HAUBERT
670	ELC1074	EDUARDO MACHADO GONDIM
671	ELC1074	EDUARDO ROSSO CARGNELUTTI
672	ELC1074	EVANDRO BOLZAN
673	ELC1074	FERNANDA CORTEZ DA ROCHA
674	ELC1074	FERNANDO QUATRIN CAMPAGNOLO
675	ELC1074	GABRIEL MACHADO LUNARDI
676	ELC1074	GABRIEL MARCHESAN

Figura 3.7: Resultado do pré-processamento no relatório dos alunos

O próximo passo foi separar os alunos em turmas de acordo com o código da disciplina que estavam matriculados. O maior desafio foi encontrar uma maneira de alterar a turma ao distinguir que o valor da próxima célula da primeira coluna é diferente da atual. A estratégia utilizada envolveu dois laços de repetição, um para executar enquanto o final do relatório não fosse sinalizado e outro enquanto não chegasse à última linha do relatório dos alunos. Dentro do

segundo laço, eram selecionados o código da turma e o nome do aluno, que então era pesquisado no arquivo *inf.ldif* para encontrar o *login* do aluno no domínio. Se ele for encontrado, o mesmo é adicionado em uma *string* que conterá todos os *logins* da turma. Caso contrário, o aluno matriculado não pertence aos cursos de Ciência de Computação ou de Sistemas de Informação e, por isso, não é possível adicionar ele automaticamente no grupo. O código referente a esta etapa pode ser visto na Figura 3.8.

```
while (!fim) {
  for (var i = inic_turma; i <= alunos.getNumRows(); i++) {
    codigo_turma = alunos.getCell(i,1).getValue();
    aluno = alunos.getCell(i, 2).getValue().toLowerCase();

    // Encontra o login do aluno
    if (inf.indexOf(aluno) > 0) {
      aluno = inf.substring(inf.indexOf(aluno) + aluno.length + 1,
                            inf.indexOf("\n", inf.indexOf(aluno) + aluno.length + 1));
      matriculados += aluno + "\n";
    }
    if (i < alunos.getNumRows() && alunos.getCell(i+1, 1).getValue() != alunos.getCell(i, 1).getValue()) {
      inic_turma = i + 1;
      break;
    }
    else if (i >= alunos.getNumRows()) {
      inic_turma = alunos.getNumRows();
      break;
    }
  }
  if (codigo_turma.search("ELC") >= 0) { // Se o código da turma contém ELC, cria o arquivo com os logins
    for (var i = 1; i <= professores.getNumRows(); i++) {
      if (professores.getCell(i,1).getValue() == codigo_turma) {
        nome_prof = professores.getCell(i,2).getValue().toLowerCase();
        if (inf.indexOf(nome_prof) > 0) {
          nome_prof = inf.substring(inf.indexOf(nome_prof) + nome_prof.length + 1,
                                    inf.indexOf("\n", inf.indexOf(nome_prof) + nome_prof.length + 1));
          if (matriculados.indexOf(nome_prof) < 0)
            matriculados += "prof:" + nome_prof + "\n";
        }
      }
    }
    DocumentApp.create(codigo_turma).setText(matriculados);
  }

  // Determina o final do while
  if (inic_turma >= alunos.getNumRows())
    fim = true;

  matriculados = "";
 }
}
```

Figura 3.8: Trecho do script que separa os alunos em turmas

Da mesma forma que alunos de outros cursos, professores de outros departamentos, como Matemática e Administração, não possuem registro no domínio *inf.ufsm.br*. Tendo isso em mente, foi decidido que seriam criados apenas grupos para as discilinas da área de Informática, ou seja, as que possuem ”ELC” no seu código. Portanto, após testar se o código é de uma disciplina desta área, procura-se o nome do professor no relatório de professores a partir do código da disciplina. Se ele for encontrado, seu *login* é adicionado na lista de matriculados, mas com a diferença de conter um prefixo ”*prof:*”. Este prefixo será utilizado no momento de criar o grupo.

Por fim, o arquivo é criado, sendo nomeado com o código da disciplina e contendo todos os matriculados e o professor. Os arquivos resultantes da execução deste script serão utilizados pela próxima parte.

3.3.2.2 Script dos Grupos

Como na primeira parte deste script, a atividade realizada como experimento utilizava funções envolvendo grupos e membros de um domínio, que seriam usados nesta etapa. No entanto, naquela atividade foi feita apenas uma busca por grupos e seus membros. Para este script, foi preciso criar grupos, adicionar membros e adicionar proprietários. Um proprietário do grupo é um membro que possui privilégios como adicionar e remover outros membros. O proprietário é o professor da turma.

Antes de começar a implementação, já era esperado que o script não conseguiria criar todos os grupos em somente uma execução, principalmente pelo número de membros a serem inseridos e o número de grupos a serem criados (35 grupos). Mesmo tendo isso em mente, procurou-se primeiramente implementar a parte de criar o grupo e inserir corretamente os alunos e professores.

Para selecionar os arquivos utiliza-se a classe *DriveApp* e a função *getFiles*. Essa função retorna um vetor de arquivos, contendo todos os arquivos dentro do Google Drive do usuário. Então, para cada arquivo é testado se ele possui o prefixo ”ELC” do código da disciplina. Se sim, abre-se o arquivo, seleciona-se o contéudo e o divide em um vetor, sendo cada posição equivalente a um *login*. Esse vetor é percorrido com um laço que verifica, para cada *login*, se ele contém o prefixo ”*prof*:”. Os que possuem são adicionados como proprietário com a função *addOwner*, e os que não possuem são adicionados como membros normais através da *addMember*. A Figura 3.9 mostra o código que realiza essa tarefa.

Na primeira execução em que funcionou corretamente, o script criou nove grupos antes de exceder o limite de tempo, e o último destes não foi completamente preenchido. Ou seja, o tempo de execução foi ultrapassado durante a criação do último grupo. Disto, surgiram mais dois problemas a serem resolvidos: elaborar um método no qual fosse possível retomar a execução sem que tente recriar grupos já criados e, também, retomar o preenchimento de grupos que foram interrompidos durante o processo.

Quase concorrentemente ao período em que esta etapa estava sendo desenvolvida, o Google Apps Script recebeu uma atualização, no dia 16 de setembro de 2013 (Google Apps

```
  while (files.hasNext()) {
    var file = files.next();
    // Salva o token antes de iniciar a criação do grupo
    ScriptProperties.setProperty("token", files.getContinuationToken());
    if (file.getName().search("ELC") == 0) { // Se o nome do arquivo começa com ELC
      disciplina = file.getName();

      // Se a execução anterior excedeu o limite de tempo, é possível que o
      // grupo não tenha todos seus membros, então seleciona ao invés de criar
      try {
        group = GroupsManager.getGroup(disciplina);
      }
      catch (ex) {
        group = GroupsManager.createGroup(disciplina, "Disciplina " + disciplina).addOwner("julioc@inf.ufsm.br");
      }

      // Lê o arquivo e cria um vetor com os logins dos membros do grupo
      arquivo = DocumentApp.openByUrl(file.getUrl());
      matriculados = arquivo.getBody().getText();
      matriculados = matriculados.split("\n");
      matriculados.length -= 1;

      // Se o tamanho do vetor e o tamanho do grupo for o mesmo, o grupo está completo
      if (matriculados.length != group.getAllMembers().length)
        for (var j in matriculados) {
          if (matriculados[j].indexOf(":") > 0)
            group.addOwner(matriculados[j].substring(matriculados[j].indexOf(":")+1, matriculados[j].length));
          else
           group.addMember(matriculados[j] + "@inf.ufsm.br");
        }
    }
  }
}
```

Figura 3.9: Trecho do script que cria os grupos e insere os membros

Script, 2013b), que adicionou, além de outras, duas funcionalidades que se tornaram úteis para contornar o problema de retomar a execução:

- *DriveApp.getContinuationToken*: esta função gera um *token*, é uma espécie de código, que, ao ser interpretado corretamente, informará em qual arquivo que a execução anterior parou;

- *DriveApp.continueFileIterator*: recebendo como parâmetro um *token*, a função retorna o arquivo que o *token* possui em seu código. Com isso, é possível continuar a iteração dos arquivos.

A implementação correta do uso destas duas funções foi um desafio, visto que, por ser muito recente, não havia exemplos de outros usuários para serem usados como base. Para guardar este *token* para a próxima execução, descobriu-se que seria necessário utilizar a classe *ScriptProperties*. Essa classe gerencia as propriedades, como mencionado na seção 2.1, definidas no menu *File*, mais especificamente as *Project Properties*.

Ao executar pela primeira vez o script, o resultado da função *getContinuationToken* deve ser difinido como uma *ScriptProperties* para cada arquivo. Na segunda execução, o *token* é lido a partir da propriedade e usa-se a função *DriveApp.continueFileIterator* ao invés de começar

novamente do início. Para resolver o segundo problema citado anteriormente, o *token* deve ser gerado antes de iniciar o processamento do arquivo em questão para que, caso a execução exceda o limite de tempo durante o processamento daquele arquivo, na próxima execução inicie-se pelo arquivo interrompido.

Corrigidos todos os problemas, o programa precisou ser executado quatro vezes para criar todos os grupos corretamente, contendo todos os alunos e o seu respectivo professor. A solução encontrada para que o programa seja reexecutado sem que seja necessário que um usuário faça isso está no próprio ambiente de desenvolvimento. No menu *Resources*, é possível encontrar, além de outras, duas opções: *Current project's triggers* e *All your triggers* (”Gatilhos do projeto atual” e ”Todos seus gatilhos”, em português).

Um gatilho, neste caso, é uma propriedade que faz o script ser executado em determinadas circunstâncias. Pode-se definir um gatilho que, por exemplo, execute em uma data específica, a cada certo número de meses, semanas, dias, horas e minutos. A opção que melhor se encaixa aqui é o gatilho de execução a cada dez minutos, já que com cinco minutos poderia resultar na reexecução antes do término da execução anterior, gerando problemas envolvendo o *token*.

Mesmo não sendo possível definir quantas vezes o gatilho irá funcionar, isso não chega a ser um problema, dado que:

- Os grupos já criados e com todos seus membros já adicionados não são sobrepostos e nem sofrem qualquer alteração. O Script dos Grupos inclusive testa se todos os membros já foram adicionados e, caso positivo, pula o grupo;

- O número de vezes que o método *setProperty(”token”, files.getContinuation-Token())* (linha 14) pode ser chamado em um período de tempo é limitado. O limite de chamadas não é fornecido especificamente, mas, se esse limite é atingido, quer dizer que muitos grupos estão sendo ”pulados” por estarem completos. E, quando é atingido, termina-se a execução do script e fornece uma mensagem de erro ”*Service invoked too many times in a short time: properties rateMax*”;

- O limite diário de tempo de execução de um gatilho para serviços Google Apps for Education é de seis horas (Google, 2013b), portanto, mesmo que o gatilho fique ativo durante um dia inteiro, ele não será executado por mais de seis horas, o que é tempo mais do que suficiente para criar todos os grupos com todos os membros.

Então, independentemente de quanto tempo a mais que o necessário o gatilho fique ativo no Script dos Grupos, não acarretará em prejuízos aos resultados.

Para melhor entender o motivo de o script exceder o limite de tempo imposto no GAS, alguns testes foram realizados. Um script simples foi criado para isso e seus resultados (tempos de execução) podem ser vistos na Figura 3.10. Os testes realizados foram os seguintes:

1. Criação de 10 grupos: a duração deste teste foi de aproximadamente seis segundos (Figura 3.9a);

2. Criação de 35 grupos (número de grupos criados pelo Script dos Grupos): a duração deste teste foi de aproximadamente 21 segundos (Figura 3.9b);

3. Criação de 10 grupos com 20 membros cada: a duração deste teste foi de aproximadamente três minutos e dois segundos (Figura 3.9c).

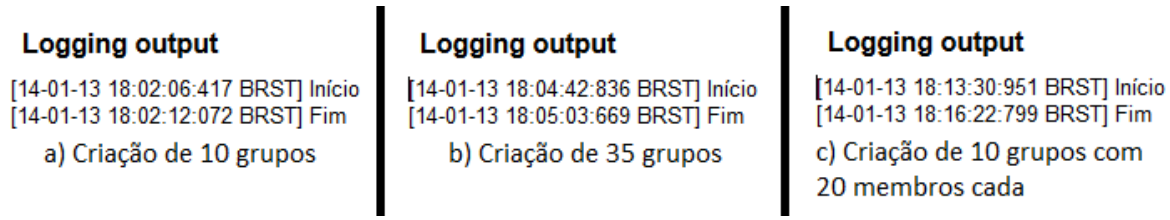


Figura 3.10: Tempos de execução dos testes realizados

Com estes resultados, podemos ver que a tarefa mais demorada envolvendo grupos no domínio neste script é a inserção de novos membros nos grupos. Usando outro script de teste, para realizar a contagem de quantos membros são adicionados ao todo durante a execução do Script dos Grupos, tem-se o número de 1.036 membros. Logo, tendo o tempo de execução do terceiro teste, que inseria 200 membros, é possivel calcular que o tempo estimado para a inserção de 1.036 é de aproximadamente 15 minutos e 42 segundos. Soma-se a isso o tempo de criação de 35 grupos. Logo, teremos 16 minutos e três segundos para criação de 35 grupos e inserção de 1.036 membros. É importante lembrar que, além desse processo de criação e inserção, o Script dos Grupos possui etapas de leitura de arquivos e busca de trechos de strings. Somando todos estes fatores, torna-se claro o porquê do Script ultrapassar o limite de tempo.

3.4 Geração de Tabelas de Horários

Nesta etapa, foi desenvolvido o segundo caso considerado. Gerar as tabelas de horários para cada semestre manualmente é uma tarefa longa e sujeita a erros que podem confundir

muitas pessoas, por isso foi escolhido como outra das tarefas para automatizar utilizando o Google Apps Script.

3.4.1 Descrição dos dados de entrada

Para obter os dados necessários, tais como nomes e códigos de disciplinas, dias e horários das aulas e código da turma, utilizou-se o mesmo arquivo usado no Item 3.3.1.3. Porém, desta vez, mais colunas da planilhas foram mantidas: código do curso; código, nome, dia da semana e horário de início da disciplina; código da turma; e o nome do professor responsável.

No entanto, este arquivo não possui a informação de a qual semestre da estrutura curricular dos cursos as disciplinas pertencem. Para contornar este problema, foi elaborada uma planilha (Figura 3.11) que serve como referência para determinar o semestre.

1	307	MTM1019, MTM1025, ELC1010, ELC1064, ELC1065, ELC1085
2	307	ELC1066, MTM1018, ELC1011, ELC1067, MTM1040, LTE1059
3	307	ELC1079, STC303, MTM1020, ELC1068, ELC117, LTE1060
4	307	MTM224, FSC135, ELC120, ELC119, ELC1069, ELC1080
5	307	ELC123, ELC1015, ELC1083, ELC131, ELC133
6	307	ELC1008, ELC1017, ELC1076, ELC1088, CAD1044
7	307	ELC1014, ELC1018, ELC408, ELC1086
8	307	ELC129, ELC1044, ELC619, ELC139, ELC1001, ELC1002, ELC1043, ELC1051, ELC1033, ELC138, ELC1089, ELC1072, DCT1031, DCT1050
1	314	CAD1002, ELC1010, ELC1064, ELC1065, ELC1075, MTM1019
2	314	CIE1002, DCT1055, ELC1011, ELC1066, ELC1067, MTM198
3	314	CAD1042, ELC117, ELC1079, ELC1069, ELC1068, STC303
4	314	CAD1043, DSP1057, ELC1080, ELC119, ELC1070, ELC1076
5	314	CTB1074, ELC133, ELC1072, ELC1071
6	314	CAD1044, ELC1017, ELC1073
7	314	ELC1077, ELC137, ELC1074
8	314	DCT1051, ELC1001, ELC1089, ELC1090, ELC1092, ELC1092, ELC1096, ELC1097, ELC1098, ELC1104, ELC1108

Figura 3.11: Planilha de referência para os semestres

Essa planilha foi feita baseando-se nas informações disponíveis no site da Informática da UFSM (Informática UFSM, 2014). A planilha contém o semestre, o código do curso e os códigos das disciplinas para o respectivo semestre. Como no oitavo semestre, as únicas disciplinas definidas são os Trabalhos de Graduação para ambos os cursos e eles não possuem horários definidos, este foi substituído pela tabela de horários das Disciplinas Complementares de Graduação (DCG's) não obrigatórias. Na Figura 3.12, estão representados a entrada e a saída do script.

3.4.2 Implementação do Script

Primeiramente, foi preciso descobrir como inserir uma tabela dentro de um documento de texto. O GAS provê classes para tratamento de tabelas, tais como *Table, TableCell* e *TableRow*. Porém, ao realizar alguns testes, estas classes pareceram estar complicando algo simples,

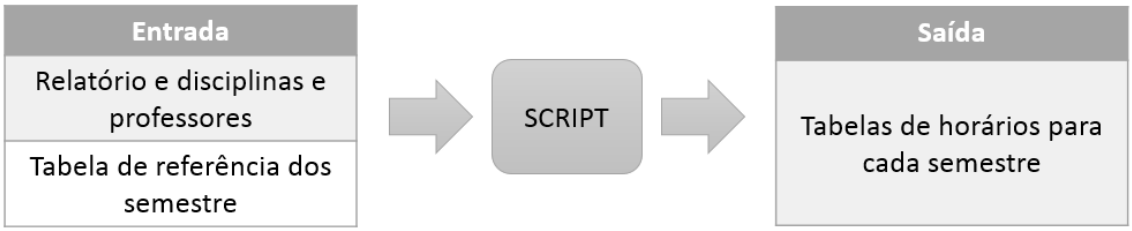


Figura 3.12: Esquema de entrada e saída do segundo caso

pois para cada célula que se desejasse inserir na tabela, era necessário criar uma nova instância da classe *TableCell* e realizar as operações de inserção de texto sobre ela. Explicar-se-á mais sobre isso no Item 4.2.4.

Na busca por alternativas, notou-se que a classe *DocumentApp* trata a tabela como uma matriz de duas dimensões. Ou seja, ao invés de criar uma variável para cada célula e linha, seria mais simples criar uma matriz. Desta forma, foi possível criar para cada curso um vetor de oito posições, equivalentes aos oito semestres, e cada posição com uma matriz de duas dimensões. Ou seja, em outras palavras, uma matriz de três dimensões. Por exemplo, o acesso a uma posição da tabela é feito desta forma: *variavel[semestre][horário da aula][dia da semana]*.

Tendo as tabelas criadas, o pré-processamento do arquivo Relatório de disciplinas e professores (Item 3.3.1.3) foi feito buscando manter as colunas necessárias, citadas no Item 3.4.1. Então, para cada linha neste arquivo, selecionou-se o código da disciplina e o seu código do curso. Assim, conseguiu-se a correspondência na tabela de referência e obteve-se o semestre da disciplina. Os outros dados são coletados a partir do Relatório do SIE e, então, a partir do semestre, do horário da aula e do dia da semana, é inserida uma *string* na tabela contendo código da disciplina e da turma. Também há uma variável que servirá como legenda para a tabela de cada semestre, contendo o código e o nome da disciplina, o código da turma e o nome do professor responsável.

A última parte é inserir as tabelas nos documentos de texto que serão a saída do script. Estes documentos (um para cada curso) contém um cabeçalho para identificar e oficializar o documento e as tabelas seguidas de suas legendas. Optou-se por deixar a inserção de imagens, como o brasão da UFSM, de forma manual, por ser mais simples do que a inserção por meio do GAS. Estas tabelas serão disponibilizadas no início de cada semestre aos alunos. Cópias serão fixadas no mural das Coordenações e, além disto, serão inseridas em pontos de fácil localização na página *www.inf.ufsm.br*

3.5 Criação de Agendas no Google Calendar

Como o principal objetivo deste caso era expandir os serviços do Google Apps utilizados durante o projeto usando o Google Calendar, a base do caso anterior é aplicada a este também. A Figura 3.13 deixa claro a similaridade entre os dados de entrada os dois casos e a diferença entre a saída.

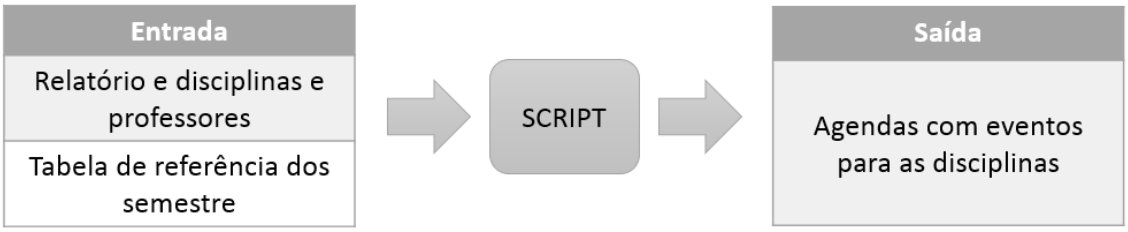


Figura 3.13: Esquema de entrada e saída do terceiro caso

A principal diferença deste caso para as tabelas de horários está na saída. Ao invés de ser adicionado algo em uma célula de uma tabela, agora será criada uma agenda para cada semestre e nelas serão adicionados eventos em série (repetitivos), representando os horários das aulas. A intenção é disponibilizar para os alunos não só um meio de verificar seus horários, como o caso anterior, mas também algo que possa lembrá-los de suas aulas e, talvez, de detalhes delas, como provas e entregas de trabalho.

3.5.1 Implementação do Script

A estrutura inicial do caso anterior foi mantida. São abertos os arquivos necessários e verificado se o pré-processamento do Relatório de disciplinas e professores já foi realizado e inicia-se a filtragem de informações. Como antes, a partir da tabela de referência descobre-se a qual semestre a matéria pertence e, então, são selecionados os dados referentes a disciplina, como horário, dia da semana, códigos de turma e de disciplina.

Com estes dados, é possível criar uma agenda ou inserir em alguma já existente. Os nomes das agendas são padronizados no formato ”Semestre X - Y”, sendo X o número do semestre e Y a sigla do curso, sendo CC para Ciência da Computação e SI para Sistemas de Informação. Para as agendas das DCG’s, os nomes são ”DCGs - Y”, com Y tendo o mesmo significado.

Como o Google Calendar permite a criação de mais de uma agenda com o mesmo nome, é necessário que, antes de mais nada, seja verificado se a agenda do semestre da disciplina que

terá seu evento criado já existe. Caso exista, ele é selecionado. O serviço Calendar do GAS, com a classe principal sendo *CalendarApp*, permite a criação de dois tipos de evento (Calendar, GAS, 2014), sendo eles *All day event*, que dura um dia inteiro, e *Event*, um evento simples, com data e horário de início e de fim.

Sabendo que as aulas possuem hora para começar e terminar, foi utilizado o evento simples. Há duas formas de eventos simples: a que acontece somente uma vez, ou seja, em um dia, e a que ocorre todas as semanas, no mesmo dia e horário. Esta segunda forma é chamada de *EventSeries*, uma série de eventos. Com ela, é possível adicionar à agenda um evento que represente a mesma disciplina e se repita toda semana, sem precisar adicionar um evento específico para cada aula.

A Figura 3.14 mostra a função *criaAgenda*, que realiza o processo de criação da agenda e inserção dos eventos.

```
function criaAgenda(semestre, hora, dia, materia, turma, curso) {
  var title;

  // Define o título da tabela
  if (semestre == 8)
    title = "DCGs - " + curso;
  else
    title = "Semestre " + semestre + " - " + curso;

  var calendarios = CalendarApp.getAllCalendars();
  var adicionar = true, calendario;

  // Verifica se a agenda já existe
  for (var i in calendarios)
    if (calendarios[i].getName() == title)
      adicionar = false;

  if (adicionar) // Se não existe, cria
    calendario = CalendarApp.createCalendar(title);
  else // Se existe, seleciona
    calendario = CalendarApp.getCalendarsByName(title)[0];

  // Adiciona a série de eventos à agenda
  var recurrence = CalendarApp.newRecurrence();
  recurrence.addWeeklyRule().times(19);
  calendario.createEventSeries(materia + " - T: " + turma, new Date(primeiro_mes +
                               " " + data(dia) + ", 2014 " + hora + ":30:00"),
    new Date(primeiro_mes + " " + data(dia) + ", 2014 " + (hora+2) + ":30:00"), recurrence);
}
```

Figura 3.14: Função *criaAgenda*

Para este caso, não encontrou-se uma opção viável para que não fosse necessário alterar as datas de início de cada semestre através de uma planilha que o script acessa para ter essas informações. Essa planilha pode ser vista na Figura 3.15. Portanto, é preciso sempre alterar o mês de início das aulas, normalmente março e agosto, e, também, os dias do mês que correspondem à primeira semana de aulas. A partir destes dias é que o script consegue adicionar as disciplinas no dia de semana, já que a data informada precisa conter o dia e o mês, e não o dia

da semana.

Mês de início (em inglês)	Dia da semana	Dia da semana das primeiras aulas (data)
March	Segunda	10
	Terça	11
	Quarta	12
	Quinta	6
	Sexta	7

Figura 3.15: Planilha com informações de datas necessárias para o script

Para tornar estas agendas disponíveis aos demais alunos e funcionários dos cursos, será inserido no site *inf.ufsm.br* um link, onde as agendas estarão incorporadas à página. Assim, todos poderão visualizar as agendas de todos os semestre, não limitando alunos que, por exemplo, estejam matriculados em cadeiras de mais de um semestre. E, como estas agendas serão públicas (qualquer pessoa que acessar a página poderá visualizar), os usuários não poderão realizar alterações nas mesmas, somente incorporar para suas agendas pessoais os eventos e, assim, poder realizar alterações, como adicionar lembretes para serem notificados por e-mail ou no seu celular.

4 RESULTADOS E DISCUSSÃO

Neste capítulo, serão expostos os resultados alcançados durante a execução do projeto. Os resultados foram além do que era esperado, visto que a proposta inicial era que fossem desenvolvidas duas aplicações, mas, como surgiu a oportunidade de se aprofundar mais nos serviços fornecidos pelo Google Apps Script, escolheu-se por desenvolver uma terceira aplicação. Também é feita uma discussão sobre dificuldades encontradas durante a implementação das aplicações. As dificuldades aqui discutidas são todas decorrentes de particularidades do Google Apps Script e exigiram mudanças na maneira de programar e de pensar do aluno.

4.1 Resultados

Além das três aplicações produzidas, este trabalho cumpriu com seu objetivo de explorar o conceito de Programação em Nuvem ao demonstrar técnicas de programação únicas em comparação com as demais já utilizadas pelo aluno no decorrer da graduação.

4.1.1 Primeiro caso

Apesar das várias dificuldades encontradas, foi possível desenvolver um aplicativo que criasse listas de e-mails para as disciplinas, composto por dois scripts: o primeiro, que cria a lista dos alunos matriculados na disciplina, e o segundo, que cria os grupos dentro do domíno *inf.ufsm.br*. Na Figura 4.1, pode-se ver alguns dos grupos criados. Essa lista fica disponível para o usuário que criou os grupos no endereço *www.groups.google.com*. Neste link, o usuário comum pode visualizar somente os grupos aos quais pertence.

Ao selecionar um grupo, é possível visualizar os membros deste grupo, com a informação adicional de quem é proprietário ou somente membro. Como pode ser visto na Figura 4.2, os únicos proprietários do grupo Disciplina ELC1008 são o criador do grupo (usuário que executou o script) e a professora da disciplina. Deve ficar claro também que, se for necessária alguma alteração no grupo, como inclusão ou remoção de membros, ela pode ser realizada pelo próprio professor, sem precisar que todos os grupos sejam recriados.

Portanto, confirmou-se que o script realiza o que foi proposto e está funcionando corretamente, podendo ser utilizado pelas Coordenações dos Cursos de Ciência da Computação e Sistemas de Informação.

Figura 4.1: Visualização dos grupos criados através do script

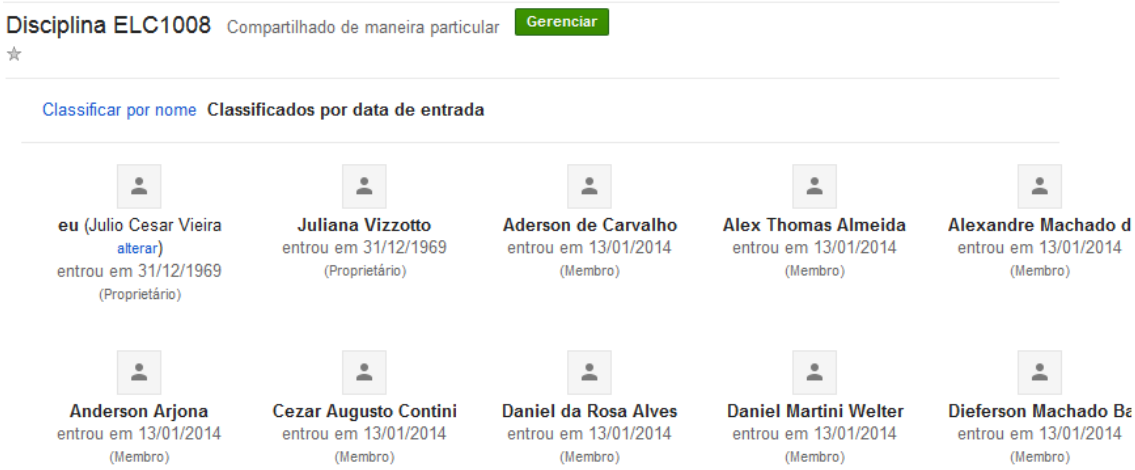


Figura 4.2: Membros do grupo Disciplina ELC1008

4.1.2 Segundo caso

Neste caso, houveram menos dificuldades encontradas no serviço GAS, mas envolveu mais problemas de programação. Foi necessário encontrar um modo de determinar o semestre de cada disciplina e também uma maneira simples de criar e inserir dados nas tabelas que representam cada semestre. A Figura 4.3 mostra o resultado da tabela de horários para o segundo

semestre do ano de 2013 do Curso de Ciência da Computação.

2º Semestre

	Segunda	Terça	Quarta	Quinta	Sexta
08:30 10:30	ELC1011 - T: CC	ELC1067 - T: CC1	ELC1011 - T: CC	ELC1067 - T: CC1	
10:30 12:30	ELC1066 - T: CC2 MTM1018 - T: 13		ELC1066 - T: CC2 MTM1040 - T: 10	MTM1018 - T: 13	MTM1040 - T: 10
14:30 16:30	ELC1067 - T: CC2	ELC1066 - T: CC1	ELC1067 - T: CC2	ELC1066 - T: CC1	
16:30 18:30	LTE1059 - T: CC		LTE1059 - T: CC		

ELC1011 - ORGANIZAÇÃO DE COMPUTADORES - Turma: CC - GIOVANI BARATTO
ELC1066 - ESTRUTURAS DE DADOS "A" - Turma: CC1 - BENHUR DE OLIVEIRA STEIN
ELC1066 - ESTRUTURAS DE DADOS "A" - Turma: CC2 - DEISE DE BRUM SACCOL
ELC1067 - LABORATÓRIO DE PROGRAMAÇÃO II - Turma: CC1 - JULIANA KAIZER VIZZOTTO
ELC1067 - LABORATÓRIO DE PROGRAMAÇÃO II - Turma: CC2 - DEISE DE BRUM SACCOL
LTE1059 - LÍNGUA INGLESA INSTRUMENTAL I - Turma: CC - SUSANA CRISTINA DOS REIS
MTM1018 - ÁLGEBRA LINEAR - Turma: 13 - DENILSON GOMES
MTM1040 - MATEMÁTICA DISCRETA "A" - Turma: 10 - LUCIANE GOBBI TONET

Figura 4.3: Tabela de horários do segundo semestre de Ciência da Computação

Através desta tabela, o aluno pode descobrir o horário, o dia e o professor da disciplina que está matriculado ou que deseja se matricular. Essas informações auxiliam os alunos no momento de montarem sua grade de horários para o semestre, evitando que haja disciplinas em horários conflitantes. São geradas tabelas para os sete primeiros semestre de ambos os cursos, já que o oitavo possui somente o Trabalho de Graduação de cada curso. Além disso, há uma tabela dos horários das Disciplinas Complementares de Graduação (DCG's).

4.1.3 Terceiro caso

Apesar de ter um propósito similar ao segundo, este caso serviu para aprofundar os conhecimentos em outros serviços do Google Apps. A Figura 4.4 mostra a agenda para as disciplinas do segundo semestre de Ciência da Computação. Se o usuário que estiver visualizando a agenda desejar copiar um evento (uma disciplina) para sua agenda, só precisa clicar no evento e em "copiar para minha agenda", como pode ser visto na Figura 4.5. Desta forma, o usuário poderá fazer alterações no evento, adicionando detalhes ou lembretes.

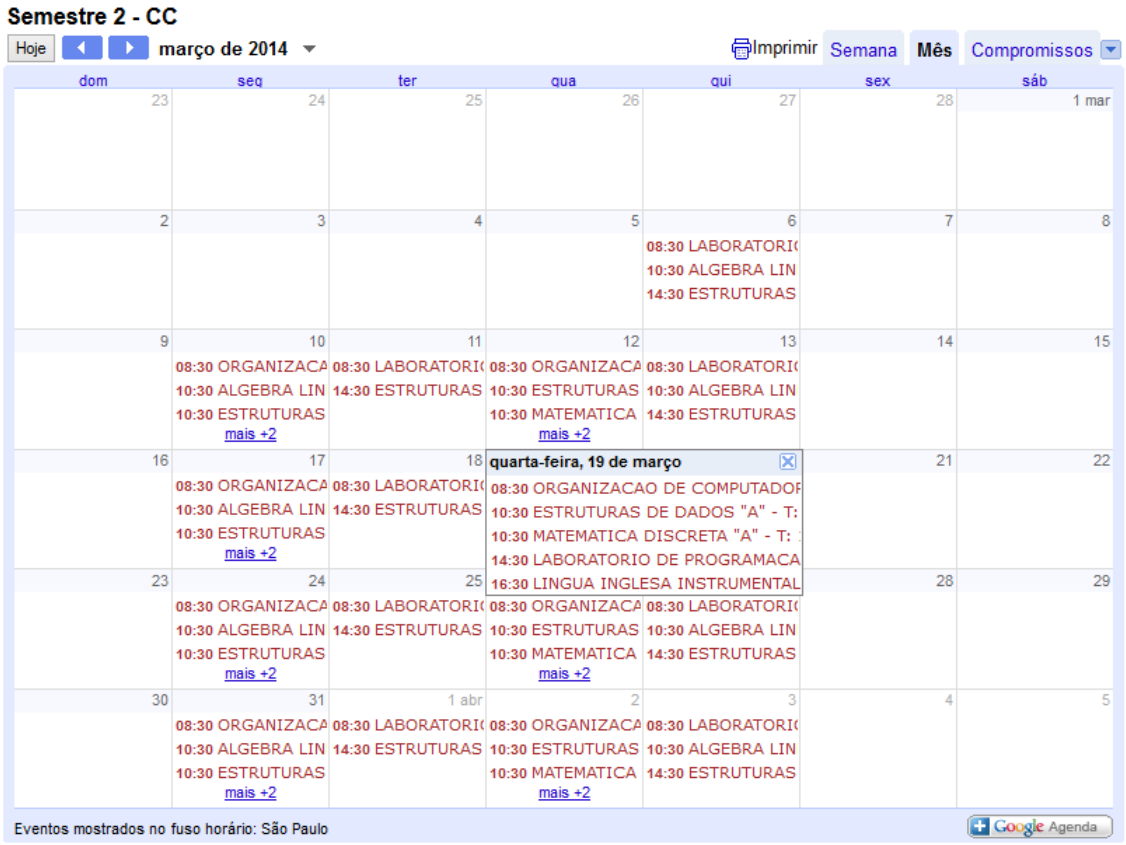


Figura 4.4: Agenda do segundo semestre de Ciência da Computação

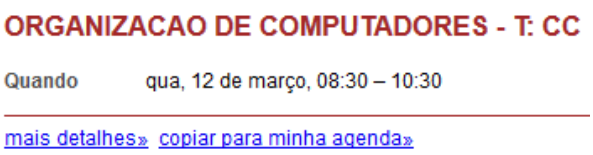


Figura 4.5: Detalhes do evento quando ele é selecionado

4.2 Discussão

4.2.1 Abrindo arquivos

A primeira particularidade sobre o Google Apps Script foi observada já na fase de experiências preliminares: o nome dos arquivos que serão lidos com os dados de entrada são absolutamente irrelevantes, devido a dois fatos: os dois modos de abrí-los utilizam apenas o URL ou uma parte dele (o ID); o Google Drive não faz distinção de nomes, podendo ter múltiplos arquivos com o mesmo nome. Não existir uma opção de abrir um arquivo através do seu nome foi considerada uma desvantagem, pois sempre que um arquivo é trocado ou atualizado, é necessário fazer alterações diretamente no código do script ou em um arquivo que é lido pelos

scripts, como foi feito neste trabalho.

4.2.2 Tempo de escrita

Outra particularidade, abordada na seção 3.2, é a diferença de tempo de escrita em diferentes tipos de arquivos. Durante o desenvolvimento do Script das Turmas, a primeira ideia foi gerar uma planilha com duas colunas. A primeira conteria o código da turma e a segunda receberia a *string* com todos os logins, separados por vírgulas. Mas, ao tentar executar o script para criar a planilha, a execução excedeu o limite de tempo imposto pelo GAS. Ao lembrar da existência de diferença de tempo de execução, o script foi alterado para o formato atual e foi executado por completo, inclusive com tempo de sobra. Assim, ficou ainda mais evidente que há diferença em escrever em planilhas ou em documentos de texto quando se trata de tempo.

4.2.3 Limite de tempo de execução

A terceira e mais destacável particularidade é o fato de existir um tempo limite de execução. Anteriormente, no Curso, o aluno sempre foi encorajado pelos professores a desenvolver algoritmos que executassem no menor tempo possível, porém nunca havia um limite de tempo. Ao programar com Google Apps Script, esse pensamento mudou. Além de ter de encontrar uma solução para os problemas de implementação, se fez necessário que as soluções fossem o mais rápidas possíveis para evitar que se atinjasse o limite de tempo. Em determinados momentos, foi preciso buscar alternativas para trechos do código, tais como qual o tipo de arquivo sendo usado para escrita e evitar percorrer laços de repetição extensos. Este trabalho incentivou o aluno a não mais somente se preocupar em implementar corretamente, mas também buscar soluções ágeis para grandes volumes de dados.

4.2.4 Criando e inserindo dados em tabelas

Criar tabelas no Google Apps Script pode ser mais complicado do que se pensa. O modo que é fornecido necessita que cada linha adicionada na tabela possua uma variável de retorno, pois é o único modo de inserir novas células, que seriam as colunas. Não foi encontrado uma forma de, utilizando os métodos e classes do GAS, definir uma tabela de dimensões *M x N*. Desta forma, buscou-se uma solução diferente: definir "manualmente" o modelo da tabela.

Inicialmente foi utilizada uma função que declarava uma variável *modelo*, definia-se o

conteúdo dela como um vetor de duas dimensões (representando o dia e o horário) e, então, retornava-se essa variável, que era inserida oito vezes em um vetor, resultando, assim, em um vetor de três dimensões: semestre, dia e horário. No entanto, ao realizar testes com o script, percebeu-se que sempre que uma tabela era alterada, todas as outras recebiam a mesma alteração.

Após verificar todo o código, concluiu-se que o problema deveria ser o retorno da variável *modelo*. A solução encontrada foi remover a variável e realizar o retorno da função diretamente com o modelo de tabela. Com isso, resolveu-se o problema. Com isso, concluiu-se que a linguagem Google Apps Script não faz distinção de variáveis inicializadas e retornadas por uma função, ao contrário de linguagens como Java e C++, que já foram utilizadas anteriormente pelo aluno e não apresentam esta característica.

4.2.5 Compartilhamento das agendas

Durante o desenvolvimento da terceira aplicação, imaginava-se que as agendas pudessem ser compartilhadas automaticamente (através do script) com todos os alunos dos cursos. Porém, ao criar a agenda, não há uma opção que permita compartilhá-la. Somente é possível convidar usuários para um evento e, como são muitos eventos e muitos usuários, este metódo torna-se inviável, até mesmo pela quantidade de convites que um usuário receberia.

Ao verificar as opções de configuração nas agendas depois de criados, através do endereço *www.google.com/calendar*, descobriu-se que é possível incorporar a agenda a uma página HTML. Com isso, o compartilhamento das agendas se torna mais simples, pois não é preciso compartilhar com usuários, e, sim, somente colocar em uma página acessível através do site *www.inf.ufsm.br*, onde ficaria disponível para todos visualizarem. Além de visualizar, torna-se possivel importar para uma agenda pessoal somente os eventos que se desejar, evitando uma poluição visual que o usuário não deseja em sua agenda.

5 CONCLUSÃO

A ideia de explorar o conceito de Programação em Nuvem e utilizar o Google Apps Script para isso mostrou-se uma escolha vantajosa, produzindo conhecimentos novos para o aluno, além de três aplicações que poderão ser utilizadas e resultarão em novas funcionalidades disponíveis a alunos e professores dos Cursos de Ciência da Computação e Sistemas de Informação.

As aplicações desenvolvidas focaram na automatização de tarefas que realizadas nos Cursos e que demandam muito tempo. A primeira, inclusive, é uma funcionalidade que ainda não era disponibilizada aos funcionários e alunos dos Cursos e que, com certeza, irá auxiliar na comunicação das turmas. No entanto, embora tenha-se buscado desenvolver códigos que pudessem ser reaproveitados com a menor quantidade de alterações possíveis entre os semestres, ainda é necessário alterar valores, como o URL dos arquivos de entrada e as datas de início das aulas em cada semestre.

É certo que o método de Programação em Nuvem evoluirá nos próximos anos, sendo capaz de oferecer serviços e tecnologias inovadoras que acompanharão a tendência de adoção de serviços em Nuvem. É provável que, nos próximos anos, mais serviços como o Google Apps Script e Windows Azure surjam, trazendo novidades para esta área recente da computação.

Recomenda-se, inclusive, às Coordenações dos Cursos que estudem possibilidades de inserção de Disciplinas Complementares de Graduação à grade curricular que apresentem e introduzam os alunos às tecnologias de Programação em Nuvem. Deste modo, seria possível trazer mais benefícios a serviços prestados pelas Coordenações e ainda preparar os alunos para possíveis novas tendências da área de computação.

Como possíveis trabalhos futuros a serem realizados com o Google Apps Script, pode-se citar dois. O primeiro seria tratar de casos específicos na criação dos grupos de e-mails (primeiro caso), como, por exemplo, quando há mais de uma turma para a mesma disciplina no semestre; atualmente, as turmas possuem um grupo único, com todos os membros e professores das turmas. Outra tarefa interessante para ser automatizada dentro dos Cursos de Ciência da Computação e Sistemas de Informação é a criação dos e-mails dos calouros. A cada semestre, é preciso que um funcionário crie 40 contas e, como é o trabalho de verificação dos nomes de calouros é manual, muitas vezes ocorrem erros de digitação.

Os scripts desenvolvidos já estão sendo testados pela Coordenação do Curso de Ciência

da Computação e é provável que sejam utilizadas já no primeiro semestre letivo de 2014, trazendo todas as vantagens e benefícios que foram pensadas durante a elaboração deste trabalho.

REFERÊNCIAS

Calendar, GAS. **Calendar Service references**. Disponível por WWW em https://developers.google.com/apps-script/reference/calendar/, acesso em 13/01/2014.

Chou, David. **Microsoft Cloud Computing Platform**. Disponível por WWW em http://www.slideshare.net/davidcchou/windows-azure-platform-2647184, acesso em 02/07/2013.

Desconhecido. **Javascript**: remove accents in strings. Disponível por WWW em http://stackoverflow.com/questions/990904/javascript-remove-accents-in-strings, acesso em 09/01/2014.

Google. **Overview of Google Apps Script**. Disponível por WWW em https://developers.google.com/apps-script/overview, acesso em 02/07/2013.

Google. **Quotas for Google Services**. Disponível por WWW em https://developers.google.com/apps-script/guides/services/quotas, acesso em 22/10/2013.

Google Apps Script. **Google Apps Script**. Disponível por WWW em http://script.google.com/, acesso em 22/10/2013.

Google Apps Script. **Release Notes, September 16, 2013**. Disponível por WWW em https://developers.google.com/apps-script/releases/, acesso em 23/09/2013.

Google Developers. **Google App Engine**. Disponível por WWW em https://developers.google.com/appengine/?hl=pt-br, acesso em 02/07/2013.

Google Support. **Introduction to Apps Script**. Disponível por WWW em https://support.google.com/sites/answer/1224162?hl=en, acesso em 02/07/2013.

Informática UFSM. **Página Web da Informática da UFSM**. Disponível por WWW em http://www.inf.ufsm.br/index, acesso em 09/01/2014.

MELL, P.; GRANCE, T. The NIST Definition of Cloud Computing. **NIST special publication**, [S.l.], v.800, n.145, p.7, 2011.

Microsoft. **Windows Azure**. Disponível por WWW em http://www.windowsazure.com/pt-br/, acesso em 02/07/2013.

Naone, Erica. **MIT Technology Review**: 10 breakthrough technologies - cloud programming. Disponível por WWW em http://www2.technologyreview.com/article/418545/tr10-cloud-programming/, acesso em 02/07/2013.

SCHEID, E. J. et al. Cloud computing with Google Apps for education: an experience report. **Journal of Applied Computing Research**, [S.l.], v.2, n.2, p.60–67, 2012.

APÊNDICES

APÊNDICE A – Código de Pré-Processamento do Arquivo *inf.ldif*

```
1  import java.io.BufferedReader;
2  import java.io.BufferedWriter;
3  import java.io.File;
4  import java.io.FileReader;
5  import java.io.FileWriter;
6  import java.util.logging.Level;
7  import java.util.logging.Logger;
8  import javax.swing.JFileChooser;
9
10 public class Interface extends javax.swing.JFrame {
11
12     public File inputFile;
13     public File outputFile;
14     public BufferedReader in;
15     public BufferedWriter out;
16
17     public Interface() {
18         initComponents();
19     }
20
21     @SuppressWarnings("unchecked")
22   private void initComponents() {
23
24         jLabel1 = new javax.swing.JLabel();
25         inputButton = new javax.swing.JButton();
26         jLabel2 = new javax.swing.JLabel();
27         outputFileName = new javax.swing.JTextField();
28         filterButton = new javax.swing.JButton();
29
30         setDefaultCloseOperation(javax.swing.WindowConstants.
               EXIT_ON_CLOSE);
31
32         jLabel1.setText("Arquivo inf.ldif");
33
34         inputButton.setText("Buscar");
35         inputButton.addActionListener(new java.awt.event.ActionListener()
               {
36             public void actionPerformed(java.awt.event.ActionEvent evt) {
37                 try {
38                     inputButtonActionPerformed(evt);
39                 } catch (Exception ex) {
40                     Logger.getLogger(Interface.class.getName()).log(Level
                           .SEVERE, null, ex);
41                 }
42             }
43         });
44
45         jLabel2.setText("Arquivo de saida");
46
47         filterButton.setText("Filtrar");
48         filterButton.addActionListener(new java.awt.event.ActionListener
               () {
49             public void actionPerformed(java.awt.event.ActionEvent evt) {
50                 try {
51                     filterButtonActionPerformed(evt);
```

```
52                  } catch (Exception ex) {
53                      Logger.getLogger(Interface.class.getName()).log(Level
                            .SEVERE, null, ex);
54                  }
55              }
56          });
57
58          javax.swing.GroupLayout layout = new javax.swing.GroupLayout(
                getContentPane());
59          getContentPane().setLayout(layout);
60          layout.setHorizontalGroup(
61              layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
                    LEADING)
62              .addGroup(layout.createSequentialGroup()
63                  .addContainerGap()
64                  .addGroup(layout.createParallelGroup(javax.swing.
                        GroupLayout.Alignment.LEADING)
65                      .addComponent(filterButton, javax.swing.GroupLayout.
                            DEFAULT_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, Short.MAX_VALUE)
66                      .addGroup(layout.createSequentialGroup()
67                          .addGroup(layout.createParallelGroup(javax.swing.
                                GroupLayout.Alignment.LEADING)
68                              .addComponent(jLabel2)
69                              .addComponent(jLabel1))
70                          .addGap(18, 18, 18)
71                          .addGroup(layout.createParallelGroup(javax.swing.
                                GroupLayout.Alignment.LEADING)
72                              .addComponent(outputFileName)
73                              .addComponent(inputButton, javax.swing.
                                    GroupLayout.DEFAULT_SIZE, 130, Short.
                                    MAX_VALUE))))
74                  .addContainerGap())
75          );
76          layout.setVerticalGroup(
77              layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
                    LEADING)
78              .addGroup(layout.createSequentialGroup()
79                  .addContainerGap()
80                  .addGroup(layout.createParallelGroup(javax.swing.
                        GroupLayout.Alignment.BASELINE)
81                      .addComponent(jLabel1)
82                      .addComponent(inputButton))
83                  .addPreferredGap(javax.swing.LayoutStyle.
                        ComponentPlacement.UNRELATED)
84                  .addGroup(layout.createParallelGroup(javax.swing.
                        GroupLayout.Alignment.BASELINE)
85                      .addComponent(jLabel2)
86                      .addComponent(outputFileName, javax.swing.GroupLayout
                            .PREFERRED_SIZE, javax.swing.GroupLayout.
                            DEFAULT_SIZE, javax.swing.GroupLayout.
                            PREFERRED_SIZE))
87                  .addPreferredGap(javax.swing.LayoutStyle.
                        ComponentPlacement.UNRELATED)
88                  .addComponent(filterButton, javax.swing.GroupLayout.
                        DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE,
                        Short.MAX_VALUE)
89                  .addContainerGap())
```

```
90          );
91
92          pack();
93      }
94
95      private void inputButtonActionPerformed(java.awt.event.ActionEvent
            evt) throws Exception {
96          JFileChooser fc = new JFileChooser();
97
98          fc.showOpenDialog(Interface.this);
99          inputFile = fc.getSelectedFile();
100         inputButton.setName(inputFile.getName());
101         in = new BufferedReader(new FileReader(inputFile));
102     }
103
104     private void filterButtonActionPerformed(java.awt.event.ActionEvent
            evt) throws Exception {
105         if (!outputFileName.getText().isEmpty()) {
106             out = new BufferedWriter(new FileWriter(outputFileName.
                    getText()));
107             String str = "";
108             while (in.ready()) {
109                 if (str.contains("cn: ")) {
110                     out.write((str.substring(str.lastIndexOf("cn: ") + 4)
                            + "\n").toLowerCase());
111                 }
112                 else if (str.contains("homeDirectory: ")) {
113                     out.write((str.substring(str.lastIndexOf("/") + 1) +
                            "\n").toLowerCase());
114                 }
115             }
116             out.close();
117         }
118     }
119
120     public static void main(String args[]) {
121         try {
122             for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing
                    .UIManager.getInstalledLookAndFeels()) {
123                 if ("Nimbus".equals(info.getName())) {
124                     javax.swing.UIManager.setLookAndFeel(info.
                            getClassName());
125                     break;
126                 }
127             }
128         } catch (ClassNotFoundException | InstantiationException |
                IllegalAccessException | javax.swing.
                UnsupportedLookAndFeelException ex) {
129             java.util.logging.Logger.getLogger(Interface.class.getName())
                    .log(java.util.logging.Level.SEVERE, null, ex);
130         }
131
132         java.awt.EventQueue.invokeLater(new Runnable() {
133             public void run() {
134                 new Interface().setVisible(true);
135             }
136         });
137     }
```

```
138
139     private javax.swing.JButton filterButton;
140     private javax.swing.JButton inputButton;
141     private javax.swing.JLabel jLabel1;
142     private javax.swing.JLabel jLabel2;
143     private javax.swing.JTextField outputFileName;
144 }
```

APÊNDICE B – Script Turmas

```
function turmas() {
  var arquivos = DriveApp.getFiles();
  var arquivo;
  var alunos, professores, urls;

  while (arquivos.hasNext()) {
    arquivo = arquivos.next();
    if(arquivo.getName().search("URL") == 0)
      break;
  }
  urls = SpreadsheetApp.openByUrl(arquivo.getUrl()).getSheets()[0].
     getDataRange();

  // Arquivo inf.ldif filtrado
  var inf = DocumentApp.openByUrl(urls.getCell(1,2).getValue()).getBody()
     .getText();// Relatório dos matriculados
  // Relatório dos matriculados
  var matric = SpreadsheetApp.openByUrl(urls.getCell(2,2).getValue());
  // Relatório dos professores por discilina
  var profs = SpreadsheetApp.openByUrl(urls.getCell(3,2).getValue());

  // Remove as colunas desnecessárias dos arquivos
  if (matric.getLastColumn() > 2) {
    matric.deleteColumns(1, 17);
    matric.deleteColumns(2, 13);
    matric.deleteRow(1);
  }
  if (profs.getLastColumn() > 2) {
    profs.deleteColumn(1);
    profs.deleteColumns(2, 9);
    profs.deleteColumns(3, 9);
    profs.deleteRow(1);
  }
  // Seleciona a planilha e depois o range das células que serão usadas
  alunos = matric.getSheets()[0].getDataRange();
  professores = profs.getSheets()[0].getDataRange();

  // Ordena os dados a partir da primeira coluna (código da discilina)
  alunos.sort(1);
  professores.sort(1);

  var inic_turma =  1;
  var fim = false;
  var matriculados = "";
  var codigo_turma, aluno, nome_prof;
  var login;
  while (!fim) {
    for (var i = inic_turma; i <= alunos.getNumRows(); i++) {
      codigo_turma = alunos.getCell(i,1).getValue();
      aluno = alunos.getCell(i, 2).getValue().toLowerCase();

      // Encontra o login do aluno
      if (inf.indexOf(aluno) > 0) {
        aluno = inf.substring(inf.indexOf(aluno) + aluno.length + 1,
                              inf.indexOf("\n", inf.indexOf(aluno) +
```

```
                                        aluno.length + 1));
54          matriculados += aluno + "\n";
55        }
56        if (i < alunos.getNumRows() && alunos.getCell(i+1, 1).getValue() !=
              alunos.getCell(i, 1).getValue()) {
57          inic_turma = i + 1;
58          break;
59        }
60        else if (i >= alunos.getNumRows()) {
61          inic_turma = alunos.getNumRows();
62          break;
63        }
64      }
65      if (codigo_turma.search("ELC") >= 0) { // Se o código da turma contém
            ELC, cria o arquivo com os logins
66        for (var i = 1; i <= professores.getNumRows(); i++) {
67          if (professores.getCell(i,1).getValue() == codigo_turma) {
68            nome_prof = professores.getCell(i,2).getValue().toLowerCase();
69            if (inf.indexOf(nome_prof) > 0) {
70              nome_prof = inf.substring(inf.indexOf(nome_prof) + nome_prof.
                    length + 1,
71                                        inf.indexOf("\n", inf.indexOf(
                                              nome_prof) + nome_prof.length +
                                               1));
72              if (matriculados.indexOf(nome_prof) < 0)
73                matriculados += "prof:" + nome_prof + "\n";
74            }
75            break;
76          }
77        }
78        DocumentApp.create(codigo_turma).setText(matriculados);
79      }
80
81      // Determina o final do while
82      if (inic_turma >= alunos.getNumRows())
83        fim = true;
84
85      matriculados = "";
86    }
87  }
```

APÊNDICE C – Script Grupos

```
function grupos() {
  var disciplina, arquivo, matriculados, group;
  var files;

  var token = ScriptProperties.getProperty("token");
  try { // Verifica se há um token salvo para continuar a iteração
    files = DriveApp.continueFileIterator(token);
  }
  catch (ex) { // Caso não haja, seleciona todos os arquivos
    files = DriveApp.getFiles();
  }

  while (files.hasNext()) {
    var file = files.next();
    // Salva o token antes de iniciar a criação do grupo
    ScriptProperties.setProperty("token", files.getContinuationToken());
    if (file.getName().search("ELC") == 0) { // Se o nome do arquivo come
        ça com ELC
      disciplina = file.getName();

      // Se a execução anterior excedeu o limite de tempo, é possível que
          o
      // grupo não tenha todos seus membros, então seleciona ao invés de
          criar
      try {
        group = GroupsManager.getGroup(disciplina);
      }
      catch (ex) {
        group = GroupsManager.createGroup(disciplina, "Disciplina " +
            disciplina).addOwner("julioc@inf.ufsm.br");
      }

      // Lê o arquivo e cria um vetor com os logins dos membros do grupo
      arquivo = DocumentApp.openByUrl(file.getUrl());
      matriculados = arquivo.getBody().getText();
      matriculados = matriculados.split("\n");
      matriculados.length -= 1;

      // Se o tamanho do vetor e o tamanho do grupo for o mesmo, o grupo
          está completo
      if (matriculados.length != group.getAllMembers().length)
        for (var j in matriculados) {
          if (matriculados[j].indexOf(":") > 0)
            group.addOwner(matriculados[j].substring(matriculados[j].
                indexOf(":")+1, matriculados[j].length));
          else
           group.addMember(matriculados[j] + "@inf.ufsm.br");
        }
    }
  }
}
```

APÊNDICE D – Script Tabelas de Horários

```
function main() {
  var header1, header2;
  var arquivos = DriveApp.getFiles();
  var arquivo, urls;

  while (arquivos.hasNext()) {
    arquivo = arquivos.next();
    if(arquivo.getName().search("URL") == 0)
      break;
  }
  urls = SpreadsheetApp.openByUrl(arquivo.getUrl()).getSheets()[0].
     getDataRange();

  var docentes = SpreadsheetApp.openByUrl(urls.getCell(4,2).getValue());
  var referencia = SpreadsheetApp.openByUrl(urls.getCell(5,2).getValue())
     ;

  var horarios_cc = DocumentApp.openByUrl(urls.getCell(6,2).getValue());
  header1 = horarios_cc.appendParagraph("Ministério da Educação\
     nUniversidade Federal de Santa Maria\nCentro de Tecnologia\
     nCoordenação do Curso de Ciência da Computação").setAlignment(
     DocumentApp.HorizontalAlignment.CENTER);

  var horarios_si = DocumentApp.openByUrl(urls.getCell(7,2).getValue());
  header2 = horarios_si.appendParagraph("Ministério da Educação\
     nUniversidade Federal de Santa Maria\nCentro de Tecnologia\
     nCoordenaçãdo Curso de  Sistemas de Informações").setAlignment(
     DocumentApp.HorizontalAlignment.CENTER);

  if (docentes.getLastColumn() > 7) {
    docentes.deleteColumns(7, 5);
    docentes.deleteColumns(8, 9);
    docentes.deleteRow(1);
  }
  var planilha = docentes.getSheets()[0];
  docentes = planilha.getDataRange();
  planilha = referencia.getSheets()[0];
  referencia = planilha.getDataRange();

  var curso, materia, codigo, turma, semestre, hora, dia, professor, temp
     ;

  // Legendas das tabelas
  var descr_sem_cc = [[""], [""], [""], [""], [""], [""], [""], [""]];
  var descr_sem_si = [[""], [""], [""], [""], [""], [""], [""], [""]];

  // Vetores que representam cada semestre
  var tabelas_cc = new Array();
  var tabelas_si = new Array();
  for (var i = 0; i < 8; i++) {
    tabelas_cc.push(cria_modelo());
    tabelas_si.push(cria_modelo());
  }

  // Forma de acesso as posições da tabela: tabela[semestre][horário][dia
```

```
    ]
47  for (var i = 0; i < docentes.getNumRows(); i++) {
48    codigo = docentes.getCell(i+1, 2).getValue(); // Seleciona o código
        da disciplina
49    curso = docentes.getCell(i+1, 1).getValue(); // Seleciona o curso da
        disciplina
50    switch (curso) {
51      case "307":
52        for (var j = 0; j < referencia.getNumRows()/2; j++) {
53          var cod_ref = referencia.getCell(j+1, 3).getValue(); //
              Seleciona as disciplinas para o semestre (j+1)
54          if (cod_ref.indexOf(codigo) >= 0) { // Verifica se a disciplina
               em "materia" está no semestre (j+1)
55            // Se está, seleciona todos os dados que serão utilizados
56            semestre = referencia.getCell(j+1, 1).getValue()-1;
57            turma = docentes.getCell(i+1, 3).getValue();
58            materia = docentes.getCell(i+1, 6).getValue(); // Seleciona o
                 nome da disciplina
59            dia = docentes.getCell(i+1, 4).getValue() - 1;
60            hora = docentes.getCell(i+1, 5).getValue();
61            hora = horario(hora); // Retorna a posição que a disciplina
                ficara em relação ao horário
62            professor = docentes.getCell(i+1, 7).getValue();
63
64            // Cria a legenda da disciplina
65            temp = codigo + " - " + materia + " - Turma: " + turma + " -
                " + professor;
66            if (descr_sem_cc[semestre].indexOf(temp) < 0)
67              descr_sem_cc[semestre] += temp + "\n";
68
69            // Adiciona na tabela
70            if (hora > -1 && dia < 6)
71              tabelas_cc[semestre][hora][dia] += codigo + " - T: " +
                   turma + "\n";
72            break;
73          }
74        }
75        break;
76      case "314":
77        for (var j = referencia.getNumRows()/2; j < referencia.getNumRows
              (); j++) {
78          var cod_ref = referencia.getCell(j+1, 3).getValue(); //
              Seleciona as disciplinas para o semestre (j+1)
79          //materia_ref = remacc(materia_ref);
80          var cod_ref = referencia.getCell(j+1, 3).getValue(); //
              Seleciona as disciplinas para o semestre (j+1)
81          if (cod_ref.indexOf(codigo) >= 0) { // Verifica se a disciplina
               em "materia" está no semestre (j+1)
82            // Se está, seleciona todos os dados que serão utilizados
83            semestre = referencia.getCell(j+1, 1).getValue()-1;
84            turma = docentes.getCell(i+1, 3).getValue();
85            materia = docentes.getCell(i+1, 6).getValue(); // Seleciona o
                 nome da disciplina
86            dia = docentes.getCell(i+1, 4).getValue() - 1;
87            hora = docentes.getCell(i+1, 5).getValue();
88            hora = horario(hora); // Retorna a posição que a disciplina
                ficara em relação ao horário
89            professor = docentes.getCell(i+1, 7).getValue();
```

```
90
91              // Cria a legenda da disciplina
92              temp = codigo + " - " + materia + " - Turma: " + turma + " -
                    " + professor;
93              if (descr_sem_si[semestre].indexOf(temp) < 0)
94                descr_sem_si[semestre] += temp + "\n";
95
96              // Adiciona na tabela
97              if (hora > -1 && dia < 6)
98                tabelas_si[semestre][hora][dia] += codigo + " - T: " +
                      turma + "\n";
99              break;
100           }
101         }
102         break;
103     }
104   }
105
106   for (var i = 0; i < 8; i++) {
107     var title_cc, title_si;
108
109     // Define o título da tabela
110     if (i == 7)
111       title_si = title_cc = "DCGs";
112     else
113       title_si = title_cc = i+1 + "º Semestre";
114
115     // Adiciona o título, a tabela e a respectiva legenda
116     horarios_cc.appendParagraph(title_cc).setAlignment(DocumentApp.
          HorizontalAlignment.CENTER).setBold(true);
117     horarios_cc.getBody().appendTable(tabelas_cc[i]).setBold(false);
118
119     if (descr_sem_cc[i] != "")
120       horarios_cc.appendParagraph(descr_sem_cc[i]);
121
122     horarios_si.appendParagraph(title_si).setAlignment(DocumentApp.
          HorizontalAlignment.CENTER).setBold(true);
123     horarios_si.getBody().appendTable(tabelas_si[i]).setBold(false);
124
125     if (descr_sem_si[i] != "")
126       horarios_si.appendParagraph(descr_sem_si[i]);
127   }
128 }
129
130 function cria_modelo() {
131   return [
132     [[""], ["Segunda"], ["Terça"], ["Quarta"], ["Quinta"], ["Sexta"]],
133     [["08:30\n10:30"], [""], [""], [""], [""], [""]],
134     [["10:30\n12:30"], [""], [""], [""], [""], [""]],
135     [["14:30\n16:30"], [""], [""], [""], [""], [""]],
136     [["16:30\n18:30"], [""], [""], [""], [""], [""]]
137   ];;
138 }
139
140 function horario(hora) {
141   switch (hora) {
142     case "08:30:00":
143       return 1;
```

```
144         break;
145       case "10:30:00":
146         return 2;
147         break;
148       case "14:30:00":
149         return 3;
150         break;
151       case"16:30:00":
152         return 4;
153         break;
154       default:
155         return -1;
156     }
157 }
```

APÊNDICE E – Script Agendas do Google Calendar

```
var datas;

function main() {
  var arquivos = DriveApp.getFiles();
  var arquivo, urls;

  while (arquivos.hasNext()) {
    arquivo = arquivos.next();
    if(arquivo.getName().search("URL") == 0)
      urls = SpreadsheetApp.openByUrl(arquivo.getUrl()).getSheets()[0].
          getDataRange();
    else if(arquivo.getName().search("Datas") == 0)
      datas = SpreadsheetApp.openByUrl(arquivo.getUrl()).getSheets()[0].
          getDataRange();
  }

  var docentes = SpreadsheetApp.openByUrl(urls.getCell(4,2).getValue());
  var referencia = SpreadsheetApp.openByUrl(urls.getCell(5,2).getValue())
      ;

  if (docentes.getLastColumn() > 7) {
    docentes.deleteColumns(7, 5);
    docentes.deleteColumns(8, 9);
    docentes.deleteRow(1);
  }
  var planilha = docentes.getSheets()[0];
  docentes = planilha.getDataRange();
  planilha = referencia.getSheets()[0];
  referencia = planilha.getDataRange();

  var curso, materia, codigo, turma, semestre, hora, dia, temp;
  for (var i = 0; i < docentes.getNumRows(); i++) {
    codigo = docentes.getCell(i+1, 2).getValue(); // Seleciona o código
        da disciplina
    curso = docentes.getCell(i+1, 1).getValue(); // Seleciona o curso da
        disciplina
    switch (curso) {
      case "307":
        for (var j = 0; j < referencia.getNumRows()/2; j++) {
          var cod_ref = referencia.getCell(j+1, 3).getValue(); //
              Seleciona as disciplinas para o semestre (j+1)
          if (cod_ref.indexOf(codigo) >= 0) { // Verifica se a disciplina
               em "materia" está no semestre (j+1)
            // Se está, seleciona todos os dados que serão utilizados
            semestre = referencia.getCell(j+1, 1).getValue();
            turma = docentes.getCell(i+1, 3).getValue();
            materia = docentes.getCell(i+1, 6).getValue(); // Seleciona o
                nome da disciplina
            dia = docentes.getCell(i+1, 4).getValue();
            hora = docentes.getCell(i+1, 5).getValue();
            hora = horario(hora); // Retorna a posição que a disciplina
                ficara em relação ao horário
            professor = docentes.getCell(i+1, 7).getValue();

            // Cria a agenda
```

```
47              if (hora > -1 && dia <= 6) {
48                cria_agenda(semestre, hora, dia + "", materia, turma, "CC")
                      ;
49              }
50              break;
51            }
52          }
53          break;
54        case "314":
55          for (var j = referencia.getNumRows()/2; j < referencia.getNumRows
                ()); j++) {
56            var cod_ref = referencia.getCell(j+1, 3).getValue(); //
                  Seleciona as disciplinas para o semestre (j+1)
57            if (cod_ref.indexOf(codigo) >= 0) { // Verifica se a disciplina
                   em "materia" está no semestre (j+1)
58              // Se está, seleciona todos os dados que serão utilizados
59              semestre = referencia.getCell(j+1, 1).getValue();
60              turma = docentes.getCell(i+1, 3).getValue();
61              materia = docentes.getCell(i+1, 6).getValue(); // Seleciona o
                    nome da disciplina
62              dia = docentes.getCell(i+1, 4).getValue();
63              hora = docentes.getCell(i+1, 5).getValue();
64              hora = horario(hora); // Retorna a posição que a disciplina
                  ficara em relação ao horário
65              professor = docentes.getCell(i+1, 7).getValue();
66
67              // Cria a agenda
68              if (hora > -1 && dia <= 6) {
69                cria_agenda(semestre, hora, dia + "", materia, turma, "SI")
                      ;
70              }
71              break;
72            }
73          }
74          break;
75      }
76    }
77  }
78
79  function cria_agenda(semestre, hora, dia, materia, turma, curso) {
80    var title;
81    var primeiro_mes = datas.getCell(2,1).getValue();
82
83    // Define o título da tabela
84    if (semestre == 8)
85      title = "DCGs - " + curso;
86    else
87      title = "Semestre " + semestre + " - " + curso;
88
89    var calendarios = CalendarApp.getAllCalendars();
90    var adicionar = true, calendario;
91
92    // Verifica se a agenda já existe
93    for (var i in calendarios)
94      if (calendarios[i].getName() == title)
95        adicionar = false;
96
97    if (adicionar) // Se não existe, cria
```

```
98      calendario = CalendarApp.createCalendar(title);
99    else // Se existe, seleciona
100     calendario = CalendarApp.getCalendarsByName(title)[0];
101
102   // Adiciona a série de eventos à agenda
103   var recurrence = CalendarApp.newRecurrence();
104   recurrence.addWeeklyRule().times(19);
105     calendario.createEventSeries(materia + " - T: " + turma, new Date(
          primeiro_mes +
106                                   " " + data(dia) + ", 2014 " + hora + "
                                        :30:00"),
107       new Date(primeiro_mes + " " + data(dia) + ", 2014 " + (hora+2) + "
            :30:00"), recurrence);
108 }
109
110 function horario(hora) {
111   switch (hora) {
112     case "08:30:00":
113       return 8;
114       break;
115     case "10:30:00":
116       return 10;
117       break;
118     case "14:30:00":
119       return 14;
120       break;
121     case"16:30:00":
122       return 16;
123       break;
124     default:
125       return -1;
126   }
127 }
128
129 function data(dia) {
130   switch (dia) {
131     case "2":
132       return datas.getCell(2,3).getValue();
133       break;
134     case "3":
135       return datas.getCell(3,3).getValue();
136       break;
137     case "4":
138       return datas.getCell(4,3).getValue();
139       break;
140     case "5":
141       return datas.getCell(5,3).getValue();
142       break;
143     case "6":
144       return datas.getCell(6,3).getValue();
145       break;
146   }
147 }
```