Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий Механики и Оптики

Факультет Информационных Технологий и Программирования

Кафедра Информационных Систем

Лабораторная работа №4
По предмету Анализ и проектирование на UML
«Итоговая модель мобильного приложения RAIC»

Исполнитель: Трофимов В.А.
Руководитель: Хлопотов М.В.
Группа: 2511

Санкт-Петербург
2014

Оглавление

Введение

В данной лабораторной работе будет подробно описана архитектура мобильного приложения с помощью диаграмм на языке UML, которое позволит пользователю использовать несколько различных собственных облачных аккаунтов как единое информационное пространство.

1. Описание объекта автоматизации

Многие пользователи сети интернет пользуются различными облачными сервисами, такими как: $Dropbox$, $Google\ Drive$, $Yandex.Disk$. Эти сервисы предоставляют пользователю небольшой (как правило от 2 ($Dropbox$) до 15 ($Google\ Drive$) ГБ, изредка 50 ($MEGA$) и 100 ($Mail.Cloud$) ГБ) объем информационного пространства, доступ к которому можно получить из любой точки мира, где присутствует доступ в Интернет. Данные сервисы позволяют осуществлять синхронизацию и контроль версий всех данных пользователя. Но довольно часто пользователи сталкиваются с проблемой нехватки предоставленного объема информационного пространства. Выходов из этой проблемы несколько – либо покупать дополнительное информационное пространство у владельцев данных сервисов (от $10\$$ в месяц и больше), либо поочередно использовать несколько облачных аккаунтов, что не совсем удобно.

Также, многим специалистам в области IT известна технология распределенного хранения данных $R.A.I.D.$ ($Redundant\ Array\ of\ Independent\ Disks$), в частности ее спецификация $RAID\ 0$, которая формирует из нескольких физических (логических) разделов единое информационное пространство, объем которого пропорционален сумме объемов разделов, на основе которых он образован. При этом управление распределением данных, хранящимися на этих разделах, инкапсулировано от вышестоящего программного обеспечения и пользователя; эту роль берет на себя физический или программный $RAID$ – контроллер.

Объединив вместе имеющуюся проблему и данную технологию, мною было предложен один из вариантов решения данной проблемы – можно объединить информационные объемы нескольких облачных аккаунтов для предоставления пользователю единого информационного пространства. Очевидно, что у данного решения может огромное количество спецификаций – от драйвера файловой системы, который полностью инкапсулирует работу с облачными аккаунтами как с обыкновенным логическим диском с файловой системой, до обычного GUI приложения. В данной модели мною описана архитектура приложения для мобильных платформ (далее – $RAIC$ ($Redundant\ Array\ of\ Independent\ Clouds$)), которое позво-

лит пользователю создавать новое хранилище, подключаться к уже ранее созданному, управлять данными, находящимися в хранилище, а также облачными аккаунтами, образующими хранилище – осуществлять подключение, отключение и перенос данных с одного облачного аккаунта на другой.

Очевидно, что при использовании технологии $RAID$ подразумевается, что к данным будет осуществляться вход только через одну точку входа – $RAID$-контроллер. Но при проектировании приложения $RAIC$ нужно учитывать то, что пользователь может пользоваться хранилищем с различных устройств, и при этом возможна ситуация, когда с двух устройств будет произведена попытка одновременно изменить данные в хранилище. Реализация многопользовательского доступа – достаточно трудоемкая задача в данном контексте, всвязи с чем архитектура моего приложения подразумевает, что пока происходит работа с хранилищем с одной копии приложения, другим копиям приложения будет отказано в доступе к данному хранилищу.

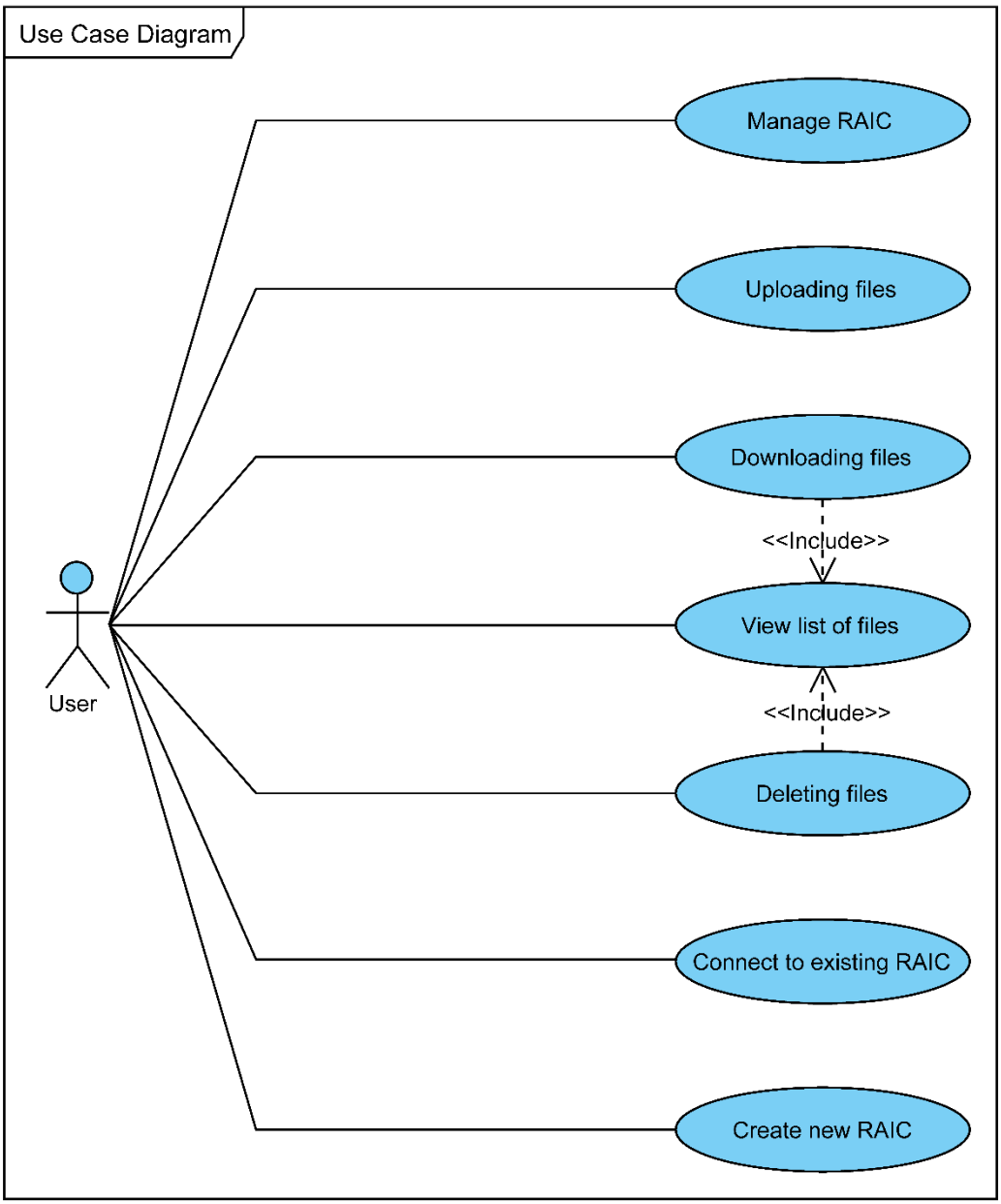
Реализация доступа с различных устройств подразумевает то, что работа приложения не должна быть привязана к какому-то конкретному мобильному устройству, т.е. все данные, как служебные, так и пользовательские, должны целиком храниться на облачных аккаунтах.

Так как API облачных сервисов предоставляют достаточно скудный набор инструментов для работы с данными на облачных аккаунтах (скачивание, загрузка, получение данных о дереве каталогов и т.п.), то реализация мьютекса ($mutex$) с помощью служебных файлов в хранилище для обеспечения доступа к нему только с одного устройства в конкретный момент времени может привести к блокировке хранилища при неудачно завершенной операции (например, внезапный обрыв соединения, перезагрузка устройства и т.п.). Архитектура приложения не указывает на какой-то конкретный метод решения данной проблемы, так как их огромное множество, и в конкретной реализации разработчик может сам выбрать подходящий алгоритм решения. Однако, на диаграмме состояний системы указано, что такое состояние при использовании данной модели является возможным. В остальном архитектура приложения подразумевает, что ситуация блокировки хранилища не возникает. Также данная модель не утверждает каких-либо конкретных

алгоритмов распределения данных между облачными аккаунтами при их изменении, так для выбора оптимального алгоритма хранения необходимо учитывать скорость доступа в интернет с конкретного устройства, скорость обмена данными с конкретными облачными сервисами, а также размеры оперируемых файлов и степень загрузки отдельных облачных аккаунтов. Модель указывает только примерную структуру хранения и обобщенную последовательность операций.

Также стоит отметить, что архитектура не подразумевает того, что один и тот же облачный аккаунт будет использоваться несколькими облачными хранилищами. Считается, что все подключаемые облачные аккаунты не наполнены ни служебной, ни пользовательской информацией.

2. Представление использования

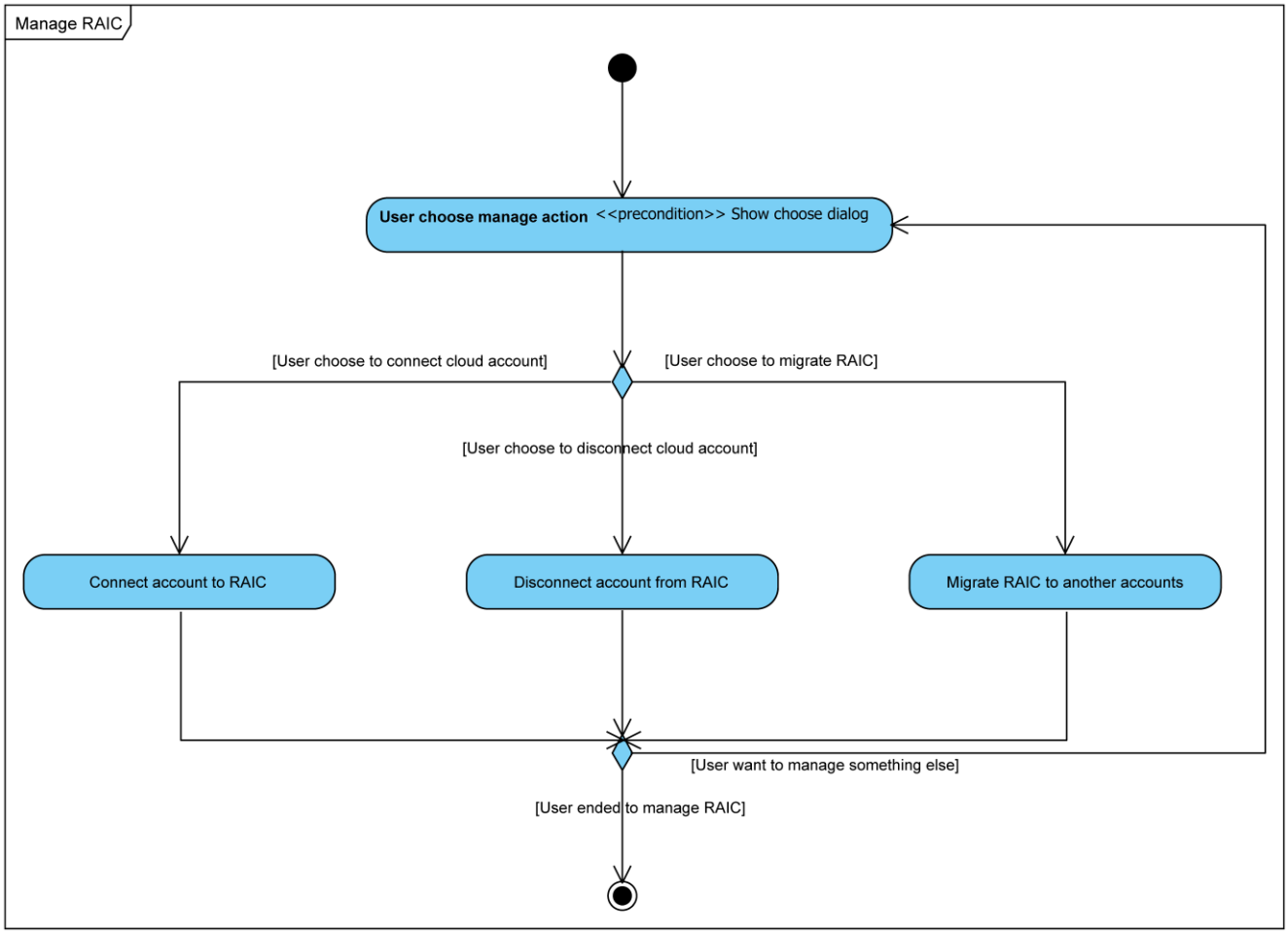


1. Диаграмма вариантов использования

Как видно из диаграммы 1, пользователю предоставляется возможно создать новое хранилище, подключиться к уже существующему хранилищу, просматривать данные, находящиеся в хранилище, скачивать и загружать туда данные, а также управлять подключенными облачными аккаунтами. Все эти пункты представляют собой главное меню приложения, откуда и осуществляется работа с хранилищем. Более подробное описание взаимодействия пользователя с приложением представлено далее, на диаграммах деятельности.

3. Представление поведения с помощью диаграмм деятельности

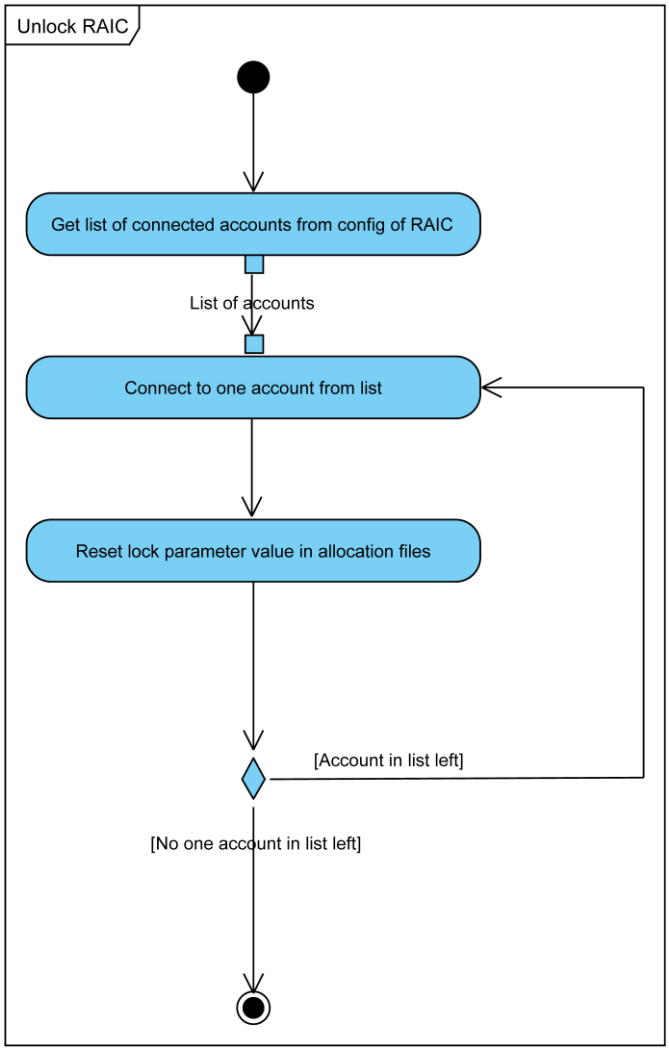
Представленные ниже диаграммы поведения описывают не только то, как пользователь взаимодействует с приолжением, но и последовательность абстрактных действий, которые будет выполнять приложение в зависимости от действий пользователя. Большинство диаграмм включают в себя составные диаграммы, это делано для декомпозиции и разгрузки диаграмм. Однако, основные диаграммы все равно получились достаточно объемными, ввиду наличия большого количества узлов решения.



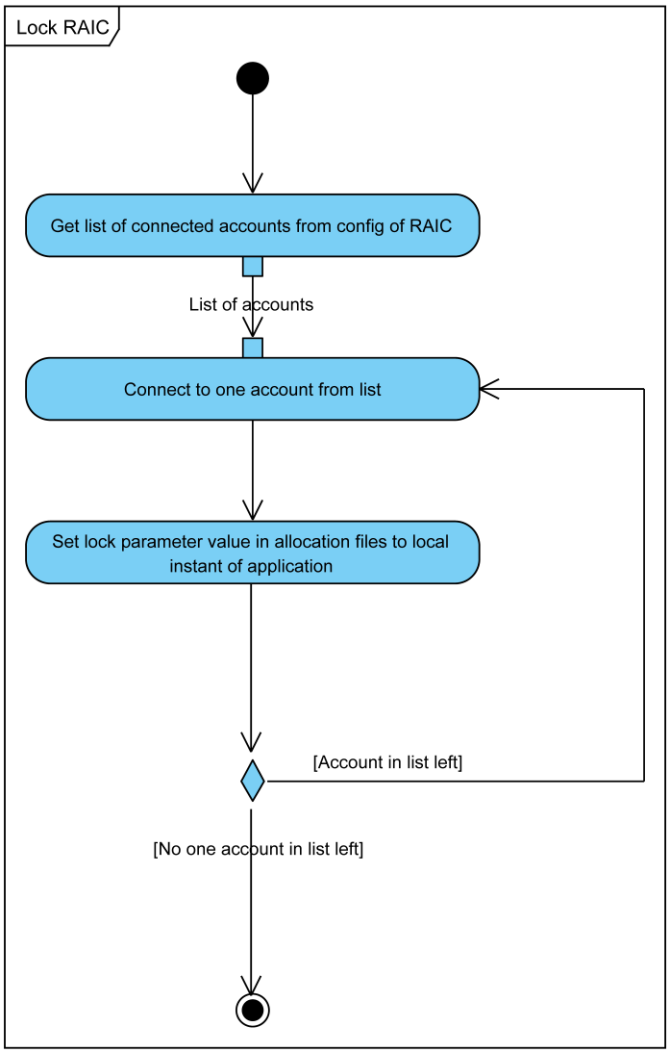
2. Диаграмма управления хранилищем

Как и описывалось ранее, пользователю предоставляется возможность подключить один или несколько аккаунтов к хранилищу, отключить аккаунты или осуществить перенос с одних облачных аккаунтов на другие, если это является возможным. После выполнения требуемой операции пользователю предлагается выполнить какую-либо другую операцию, либо вернуться в главное меню.

Ранее уже упоминалось о мьютексной системе, осуществляющей доступ только одного устройства к конкретному хранилищу в один и тот же момент времени. Для этого, при начале работы с хранилищем, в служебных файлах на всех облачных аккаунтах, образующих хранилище, параметр $lock$ устанавливается равным 1. После окончания всех операций значение этого параметра устанавливается равным 0. Если при начале работы выясняется, что параметр уже равен 1, то это означает что с хранилищем происходит взаимодействие с другого устройства, и операция отменяется. Ниже представлены 2 диаграммы деятельности, которые отражают то, как происходит удачная блокировка и разблокировка хранилища.

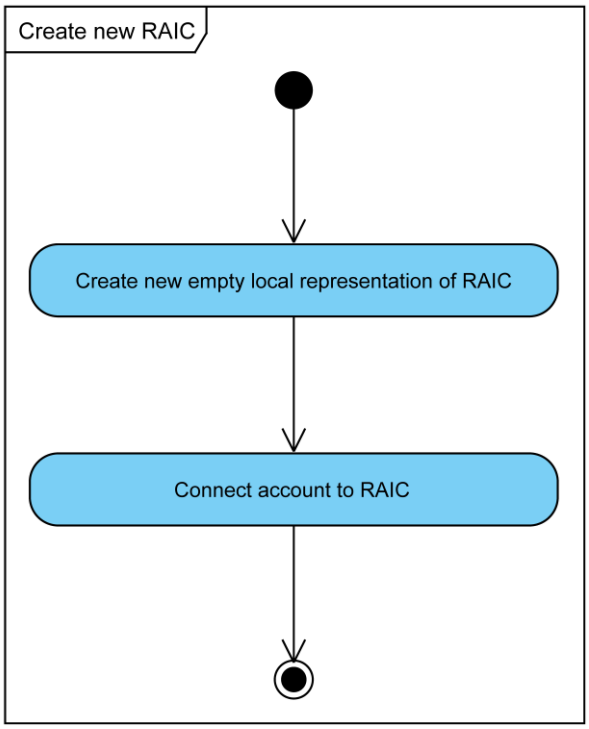


4. Диаграмма разблокировки хранилища



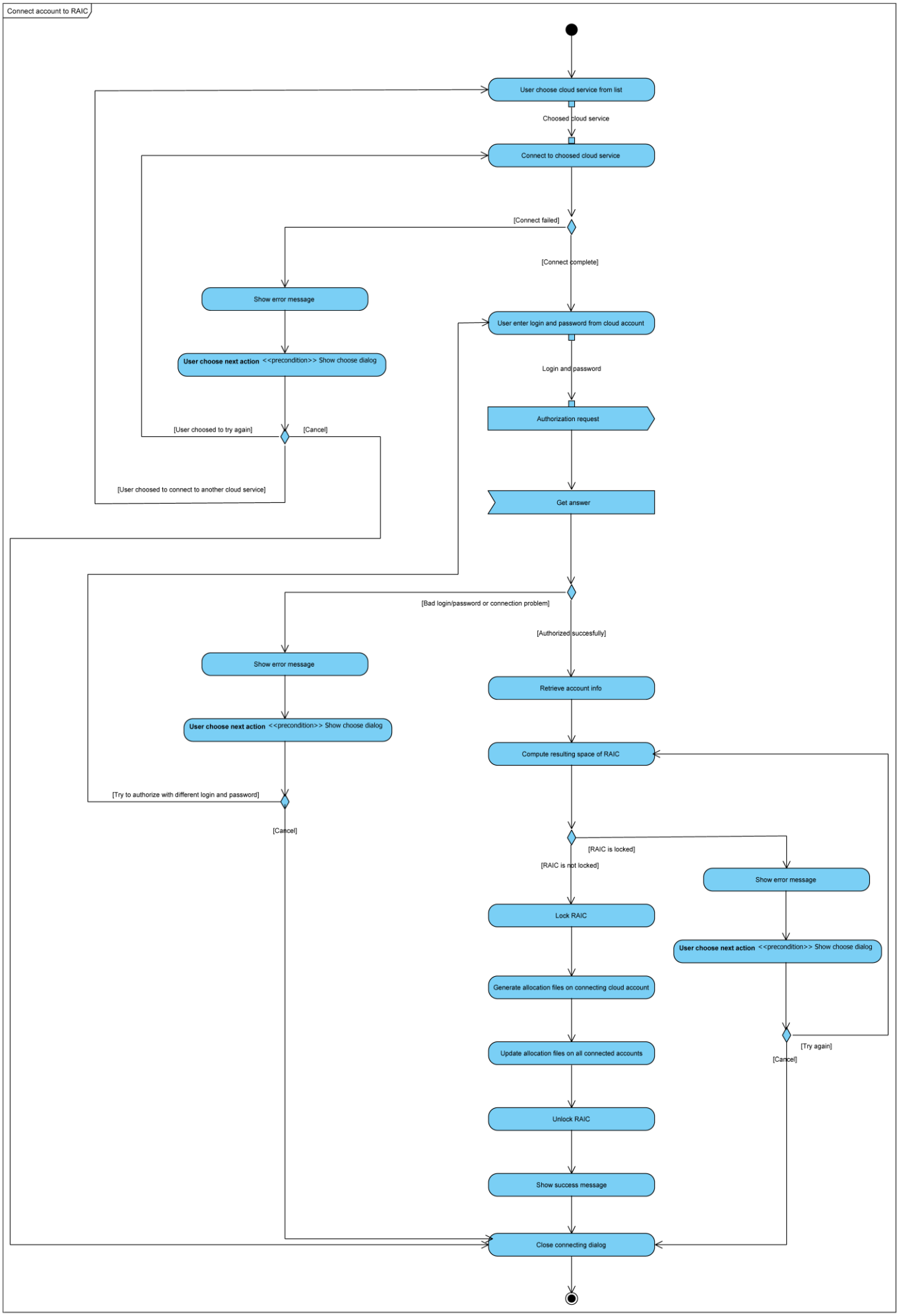
3. Диаграмма блокировки хранилища

Создание нового хранилище подразумевает создание пустого локального представления хранилища, а затем подключение хотя бы одного облачного аккаунта к системе. Данный процесс наглядно продемонстрирован на диаграмме 5, в которой действие подключения облачного аккаунта является составным.

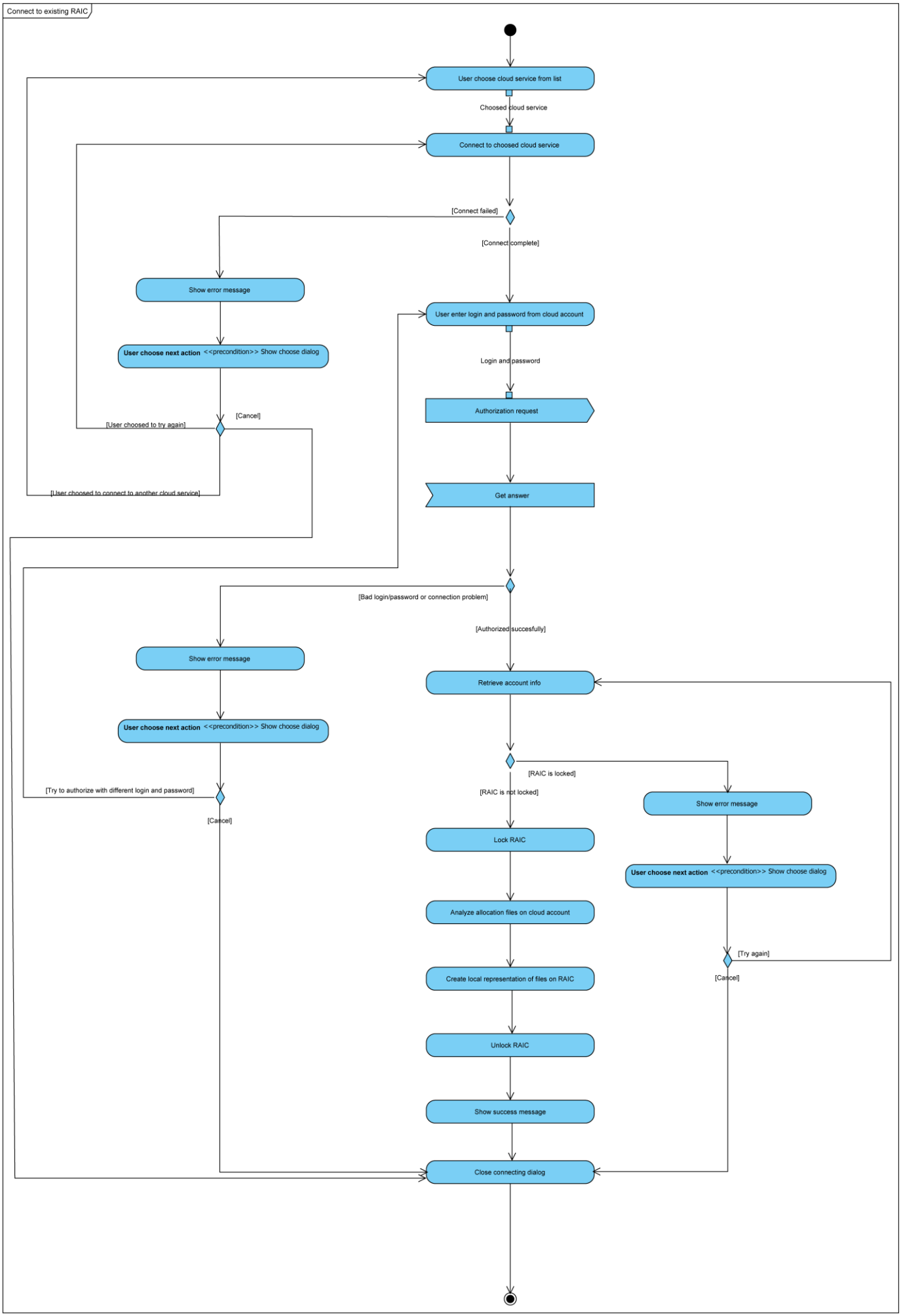


5. Диаграмма создания нового хранилища

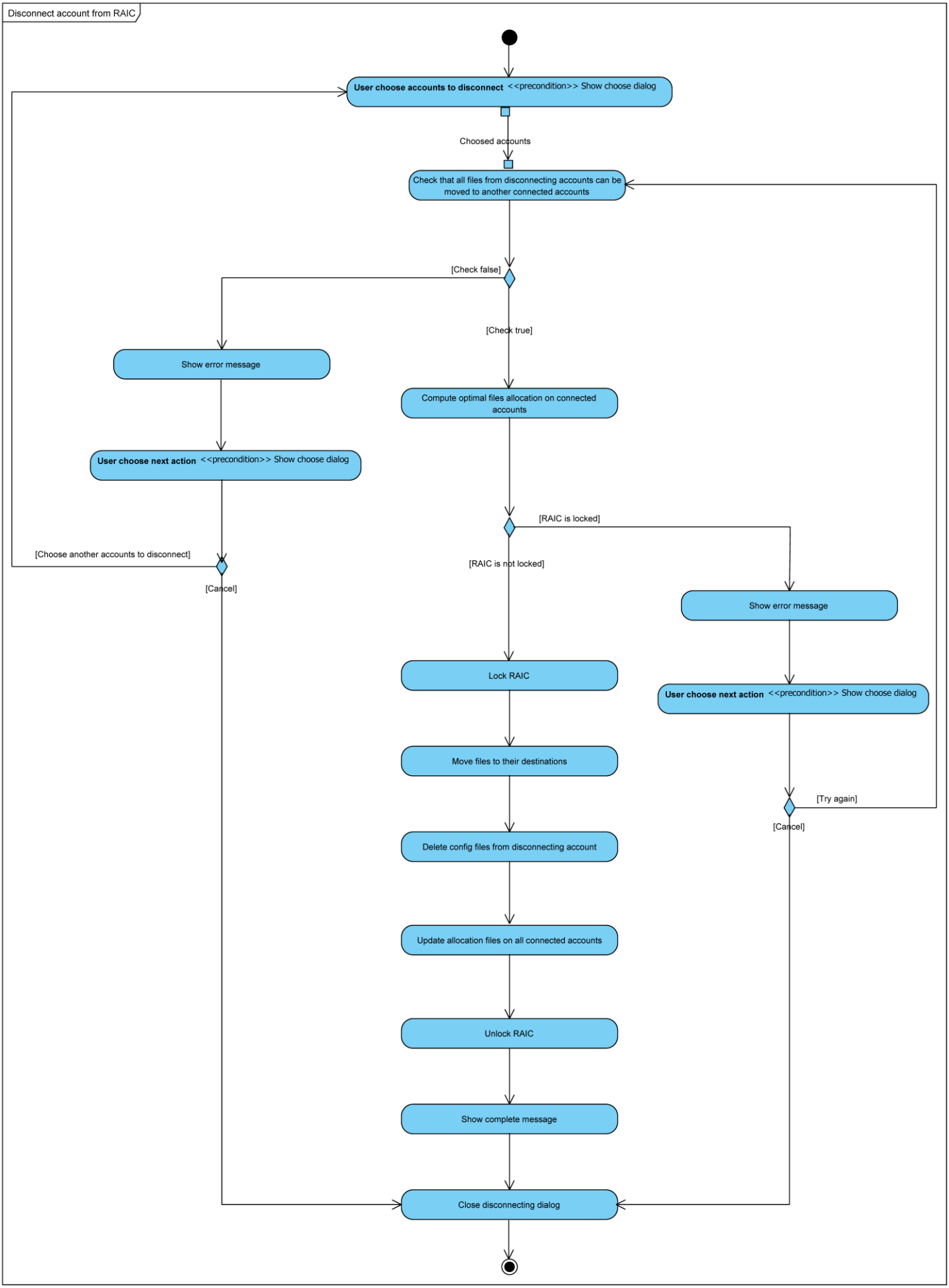
Далее следуют основные диаграммы деятельности, они достаточно большие ввиду сложности их разгрузки. Как видно, для каждого действия требуется обмен данными с облачными аккаунтами, а точнее с $allocation\ files$, расположенных на них. О том, что это такое, будет рассказано в разделе, посвященном представлению структуры. Диаграммы последовательности достаточно информативны, поэтому дополнительные пояснения вносить будет излишним.



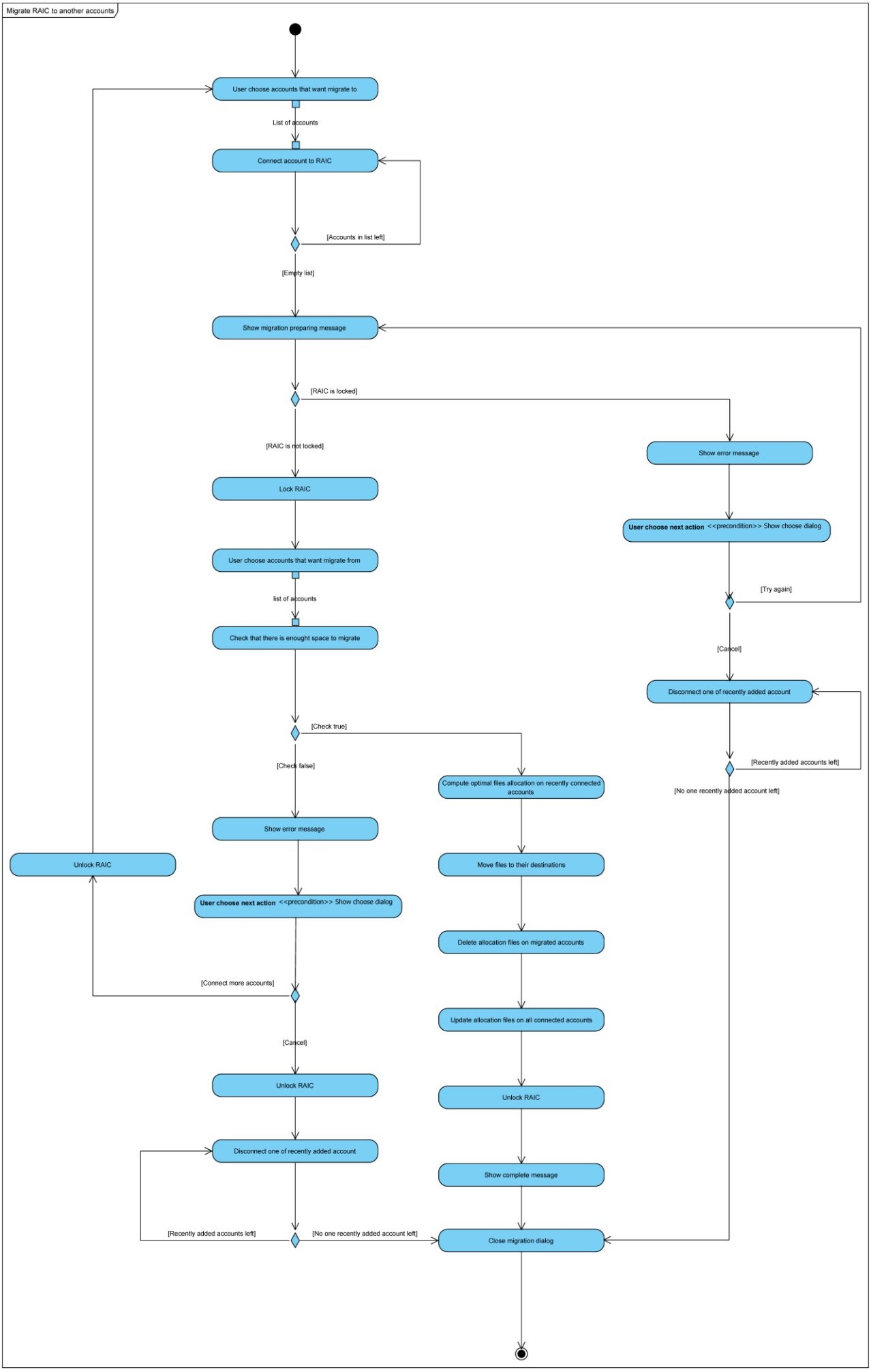
6. Диаграмма подключения облачного аккаунта к хранилищу



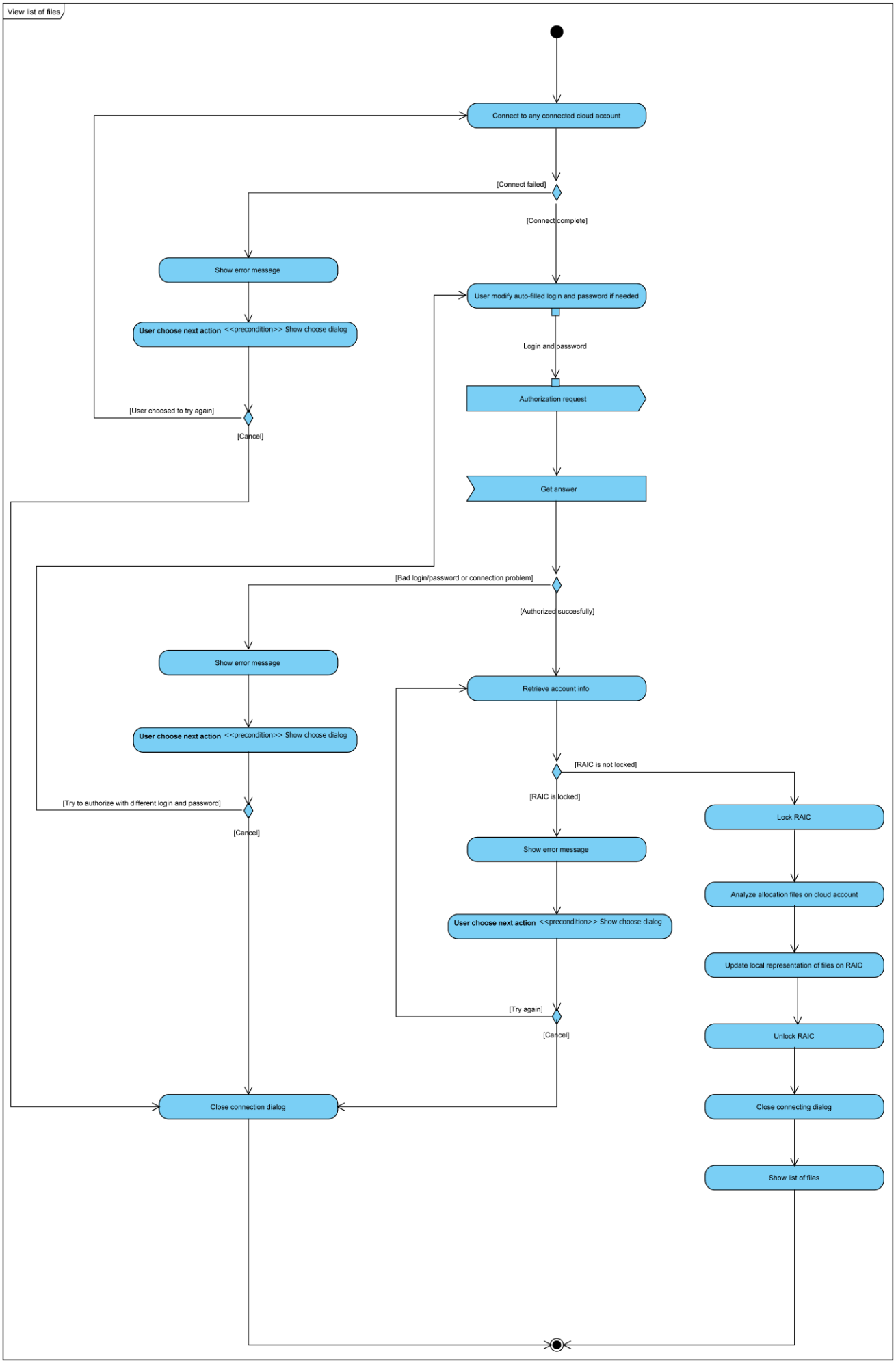
7. Диаграмма подключения к существующему хранилищу



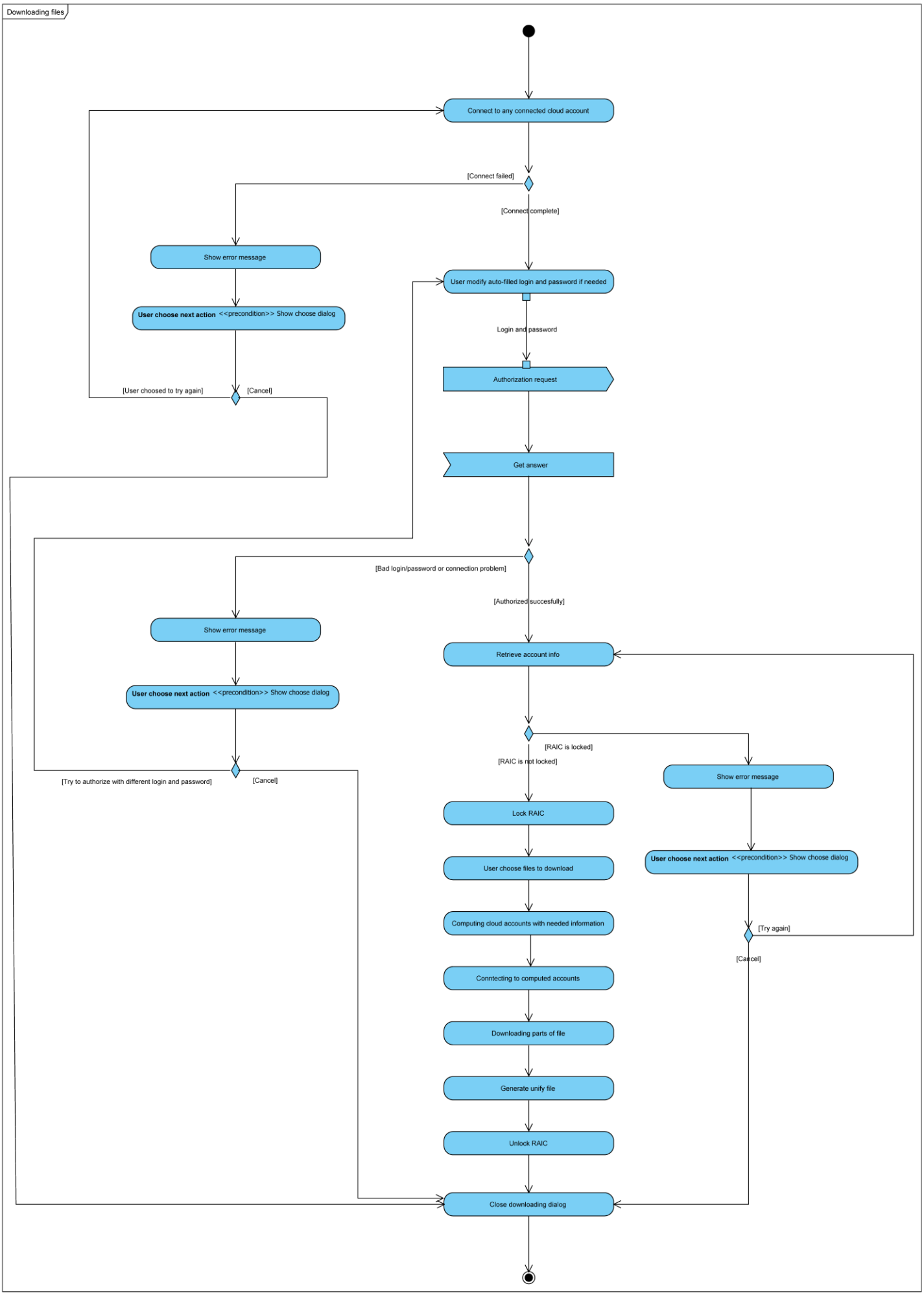
8. Диаграмма отключения облачного аккаунта от хранилища



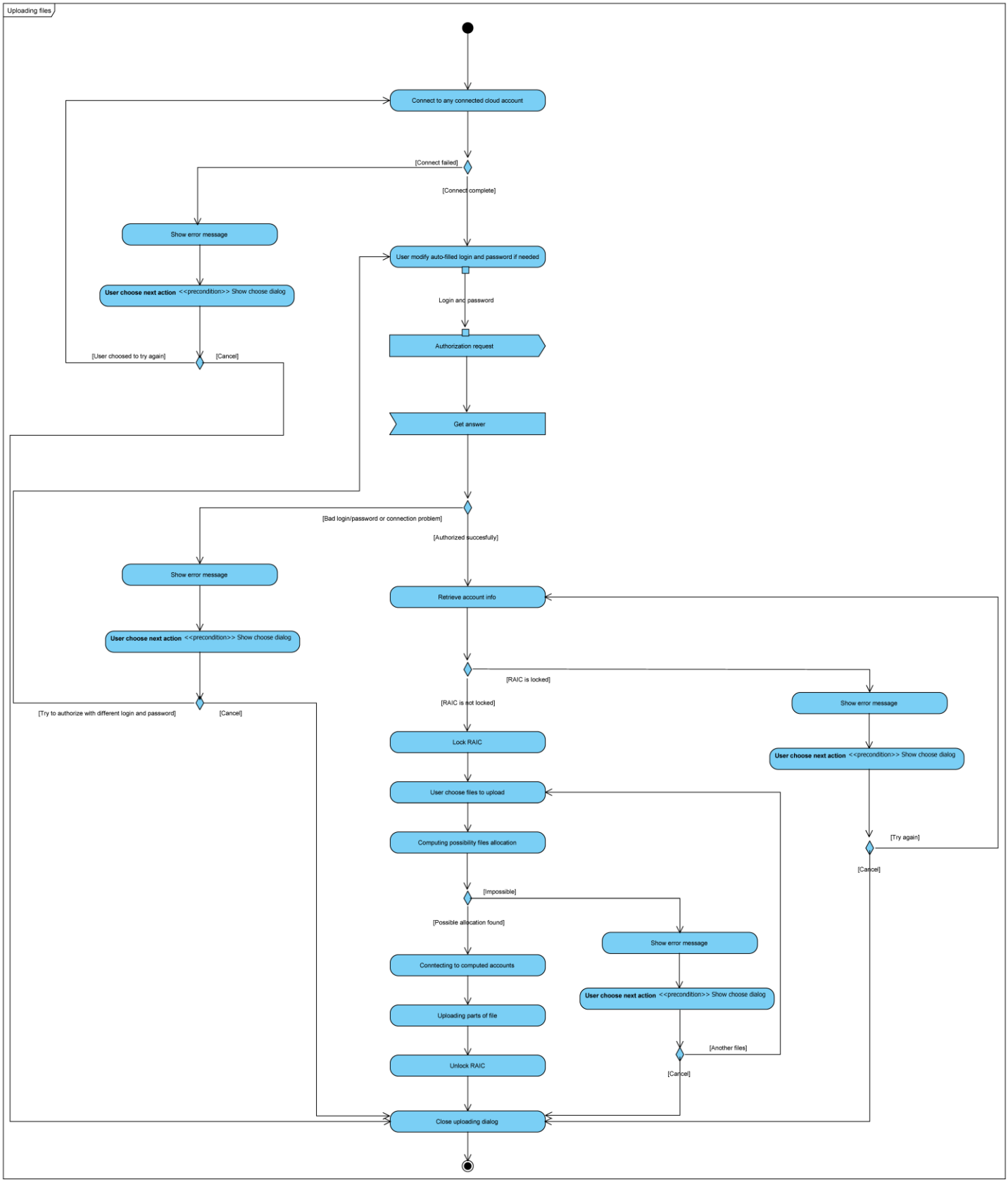
9. Диаграмма переноса данных с одних облачных аккаунтов на другие



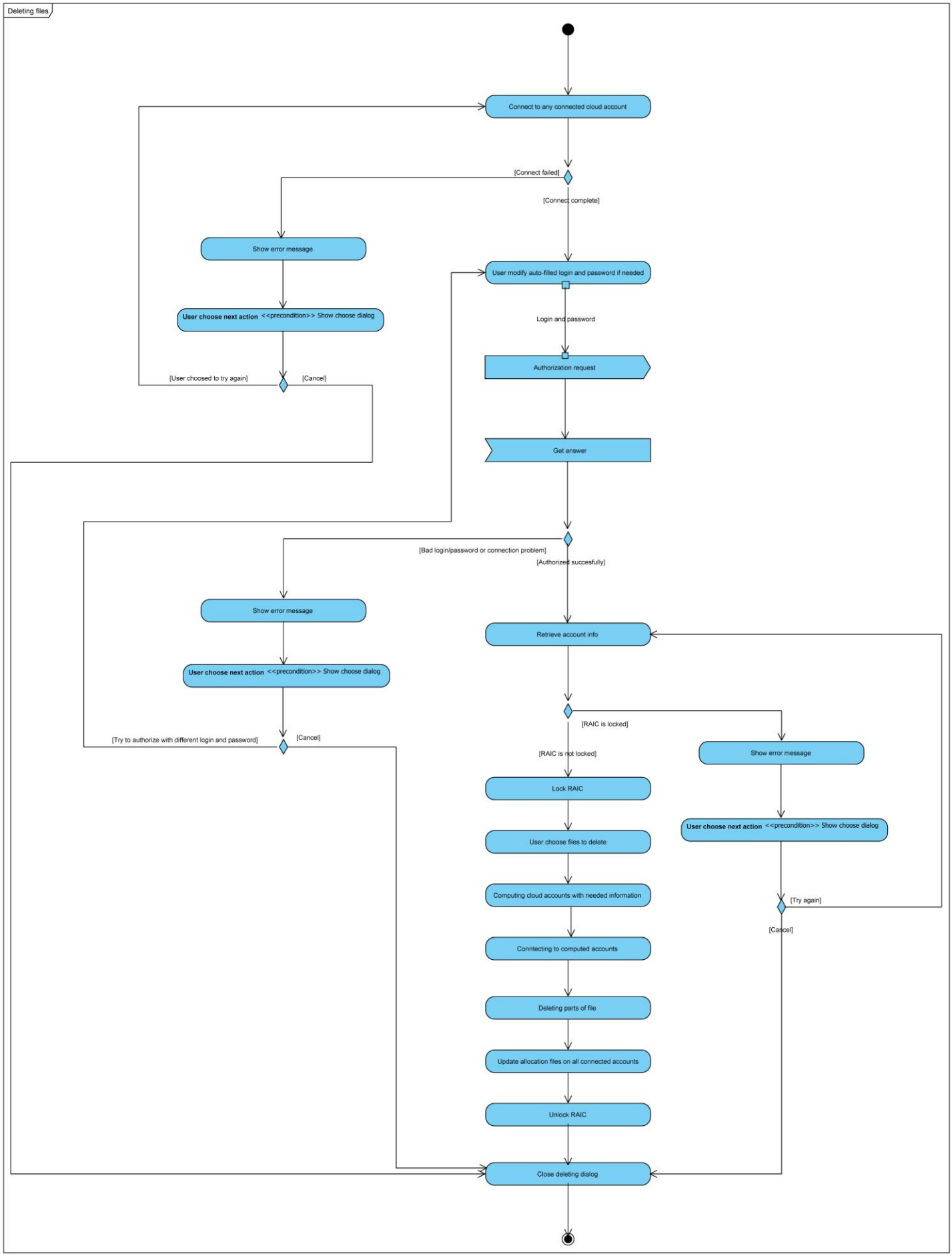
10. Диаграмма просмотра списка файлов



11. Диаграмма скачивания файлов



12. Диаграмма загрузки файлов

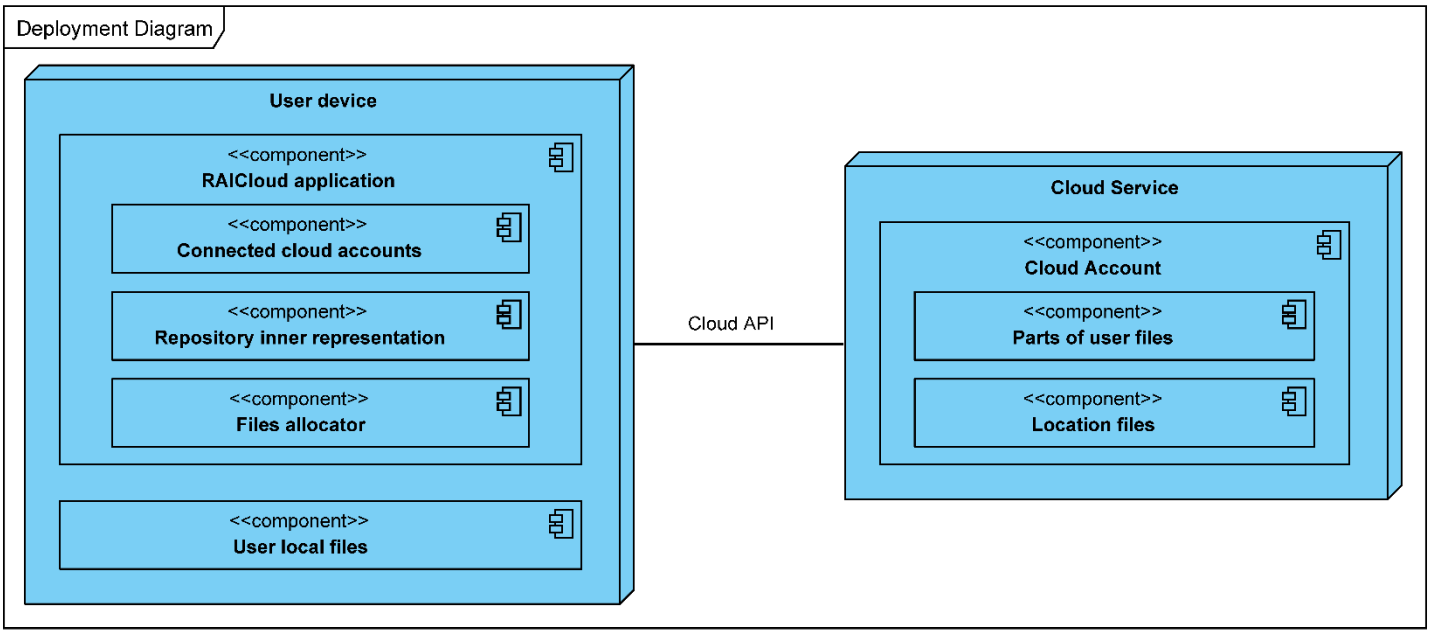


13. Диаграмма удаления файлов

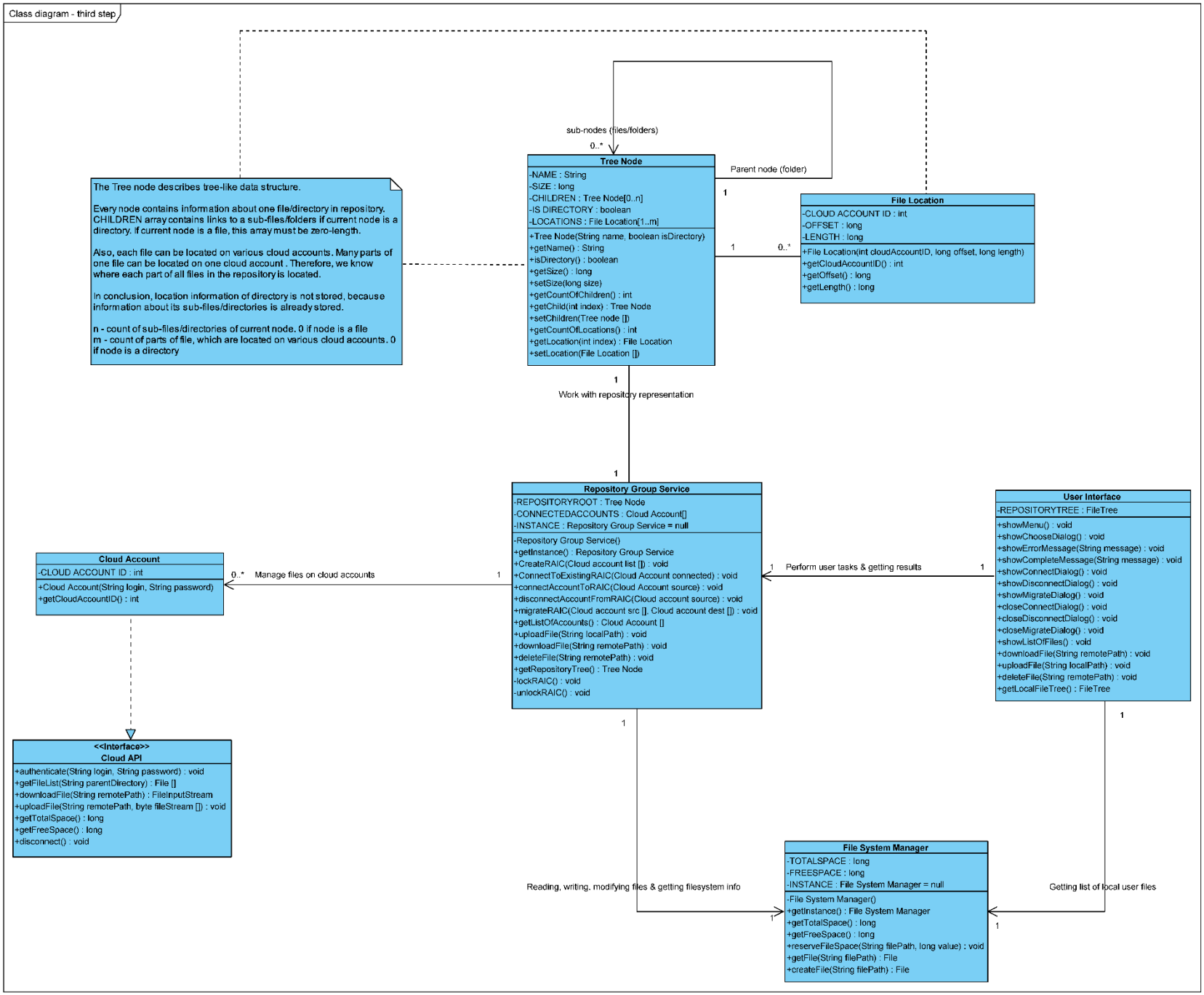
4. Представление структуры

Архитектуру приложения с точки зрения программных компонентов хорошо описывает представленная диаграмма классов 15. Внимание стоит обратить на структуры хранения данных на облачных аккаунтах и их представления в памяти мобильного устройства.

Для того, чтобы подключиться к хранилищу, пользователю достаточно авторизоваться в любом из подключенных к системе облачных аккаунтов, на каждом из которых хранится информация обо всех подключенных к хранилищу аккаунтах. Также на каждом облачном аккаунте хранится информация обо всех файлах, хранящихся в системе в виде древовидной структуры, образованной экземплярами класса $Tree\ Node$. С каждым элементом дерева, не являющимся каталогом, ассоциирован набор экземпляров класса $File\ Location$, которые хранят информацию о части файла, хранимой на конкретном облачном аккаунте. Данные хранятся в некотором сериализованном виде как $Allocation\ Files$. При обращении приложения к хранилищу, данные считываются с одного из подключенных аккаунтов и древовидная структура восстанавливается в оперативную память устройства. При завершении операции над хранилищем все изменения дерева записываются в $Allocation\ Files$ на всех облачных аккаунтах. Для более наглядной иллюстрации представлена диаграмма развертывания 14, которая также является и контекстной диаграммой.

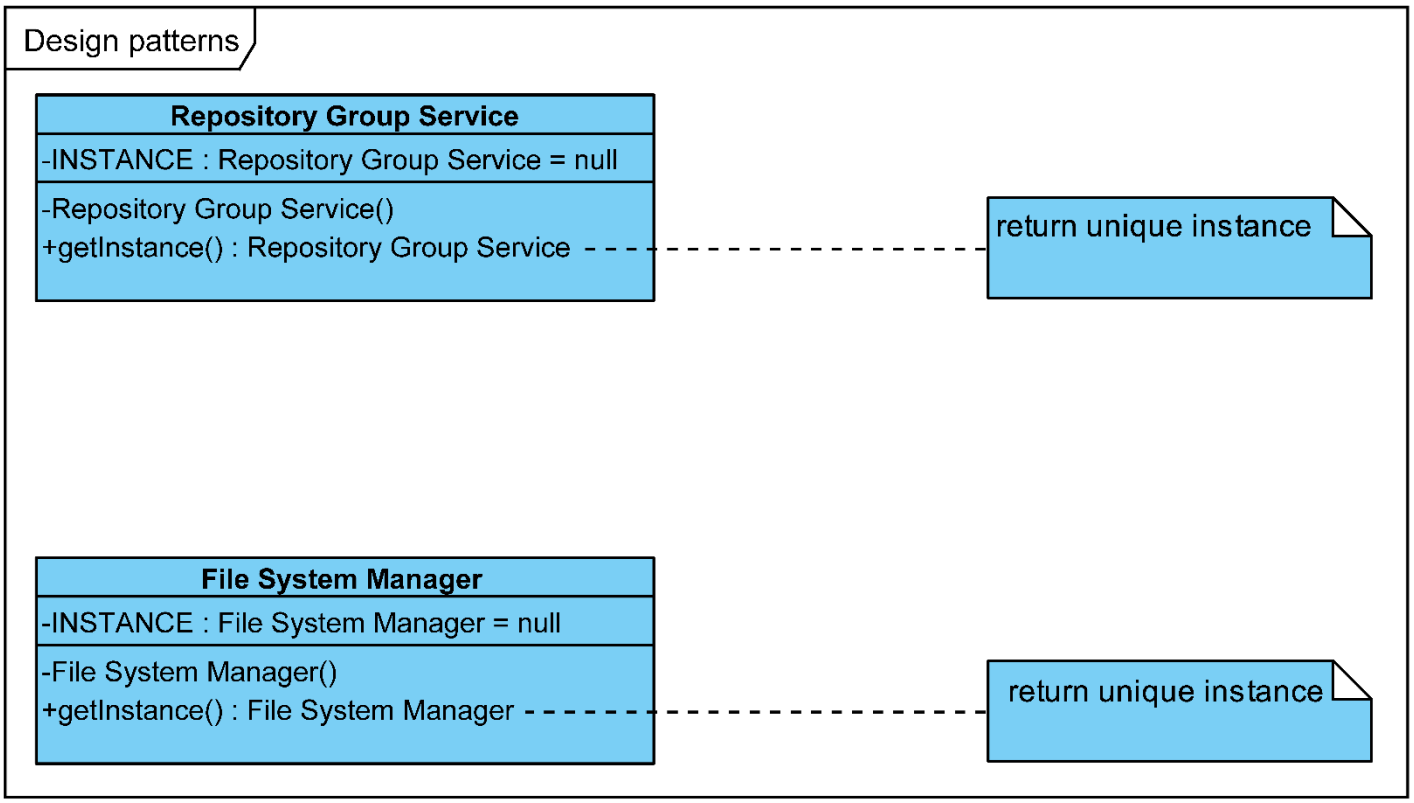


14. Диаграмма развертывания (контекстная)



15. Диаграмма классов

Для еще более наглядного отображения архитектуры представлена упрощенная диаграмма классов 16, отображающая шаблоны проектирования, которые имеет смысл использовать в данном приложении.

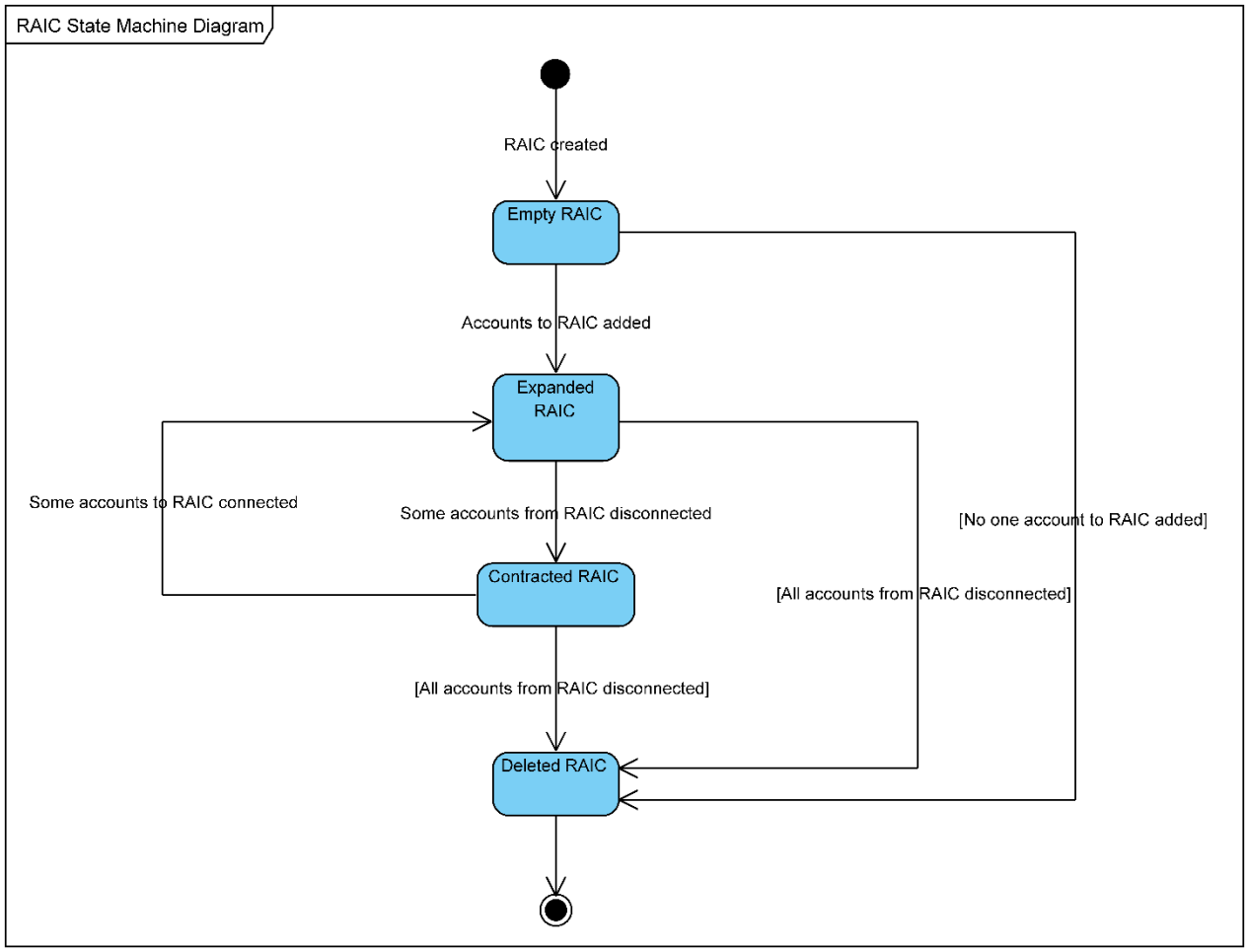


16. Шаблоны проектирования

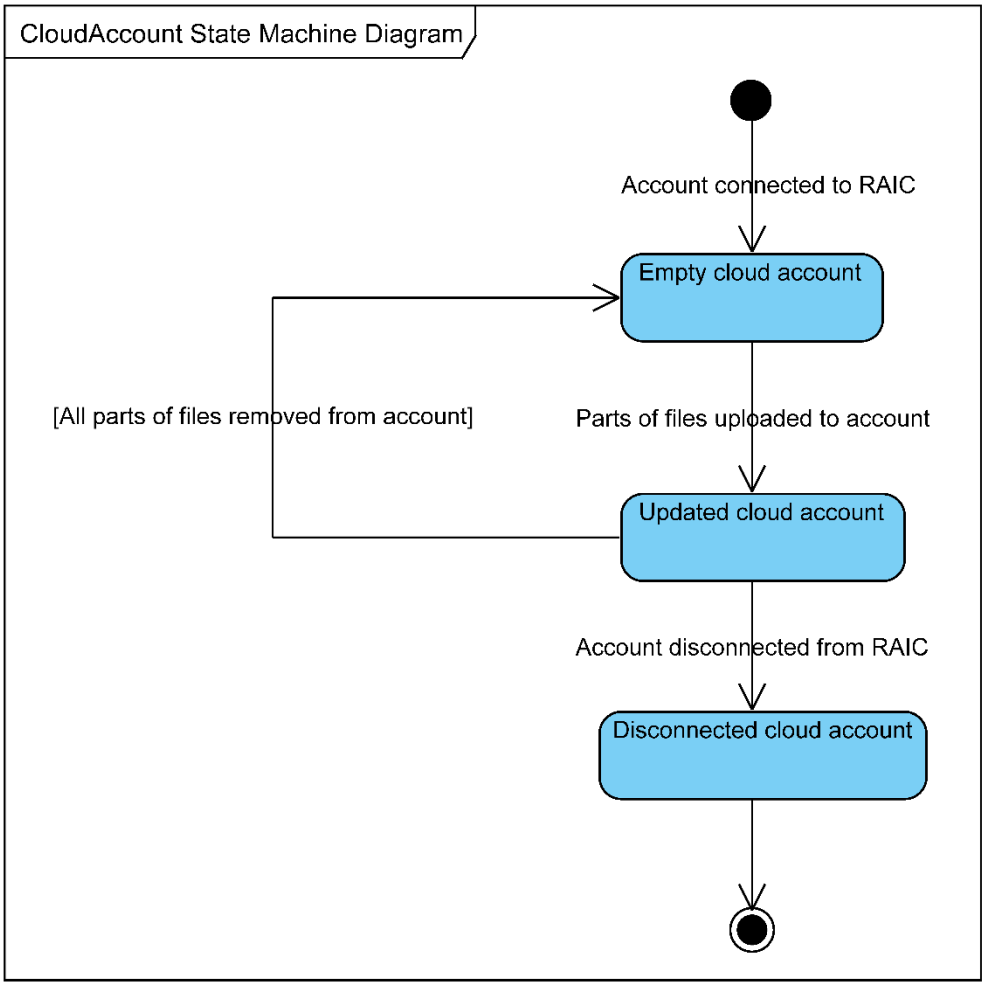
5. Представление поведения

В данном разделе будет более полно описана архитектура поведения приложения, представлены диаграммы состояний и последовательности.

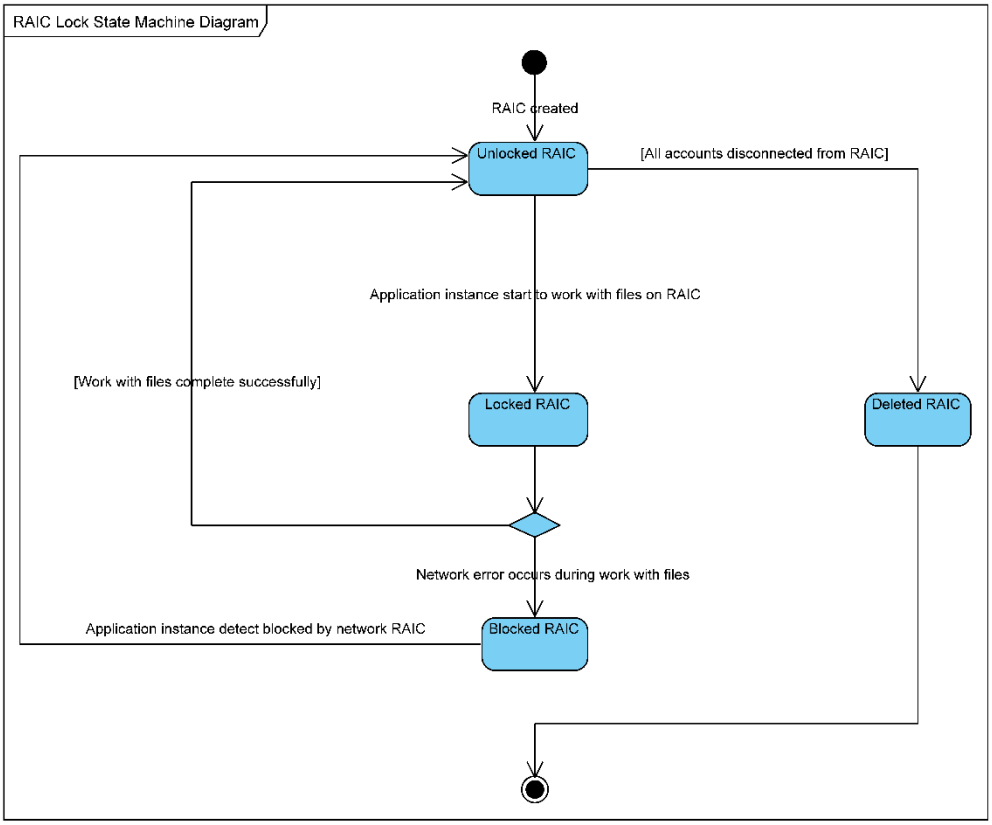
Как упоминалось ранее, логика работы приложения подразумевает, что может возникнуть ситуация блокировки хранилища в результате сетевой или другой аналогичной ошибки, что отражено на диаграмме состояний 19, одна в данной модели считается, что данная ситуация не имеет места быть, разработчик может сам взять на себя проектирование механизма разрешения блокировок, если это потребуется. Диаграмма состояний 17 описывает переходы между возможными состояниями всего хранилища в целом с точки зрения его наполненности облачными аккаунтами. Диаграмма 18 описывает возможные состояния облачного аккаунта в системе, также с точки зрения его наполненности. Диаграмма 19 описывает возможные состояния хранилища с точки зрения установленных блокировок при различных условиях. Диаграмма 20 описывает возможные состояния пользовательского интерфейса приложения.



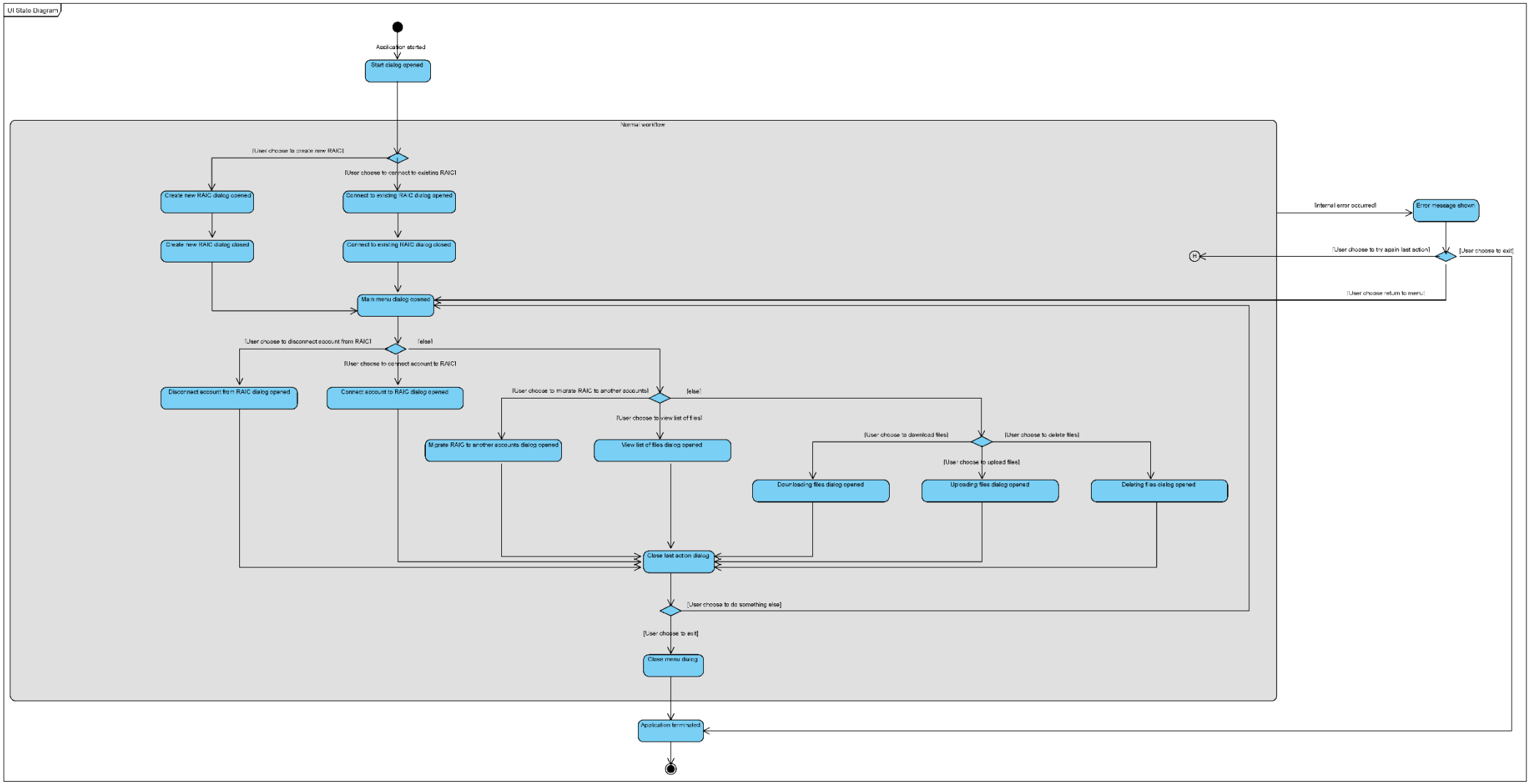
17. Диаграмма состояний хранилища с точки зрения наполненности облачными аккаунтами



18. Диаграмма состояний облачного аккаунта с точки зрения наполненности данными

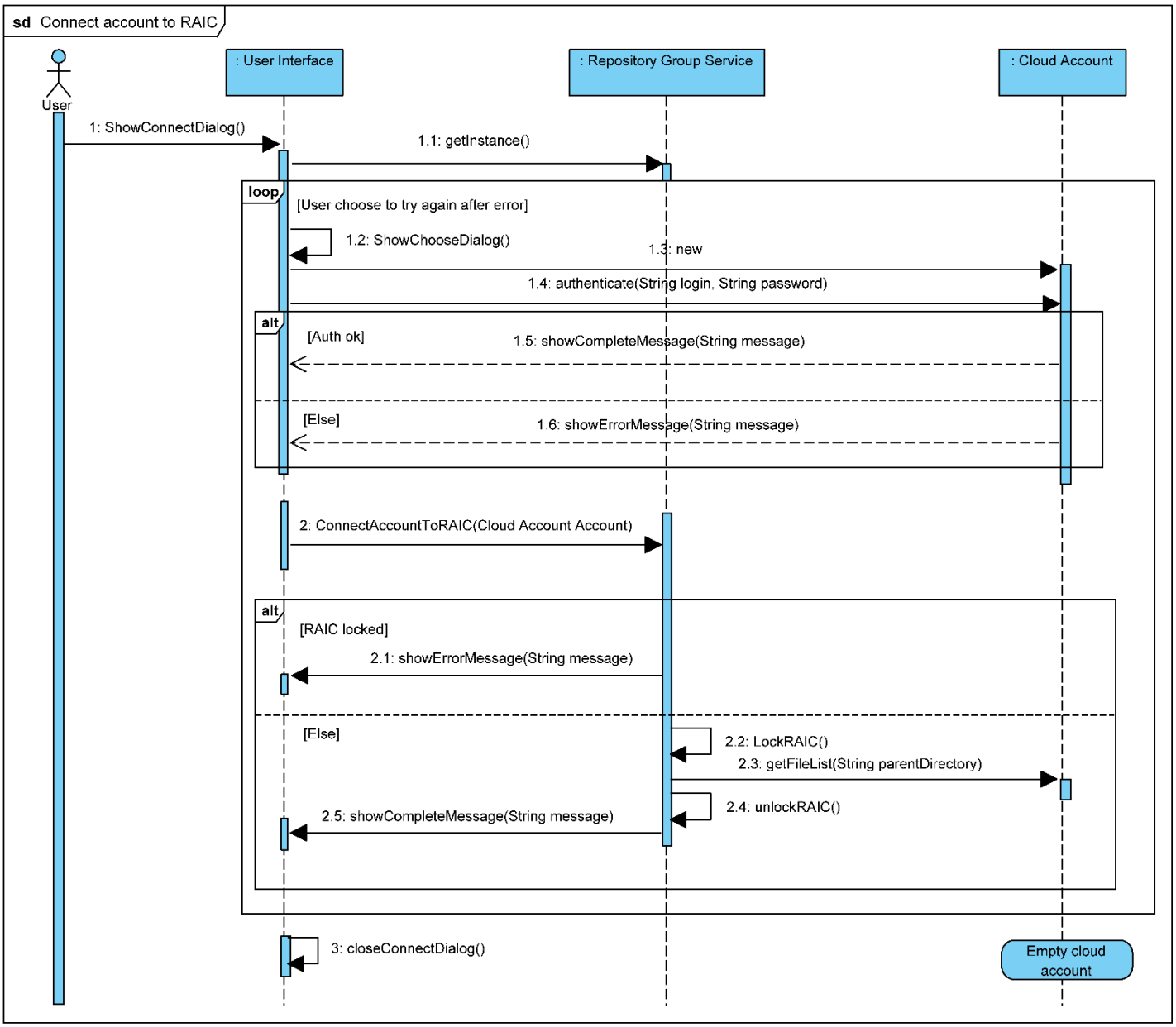


19. Диаграмма состояний хранилища с точки зрения блокировок

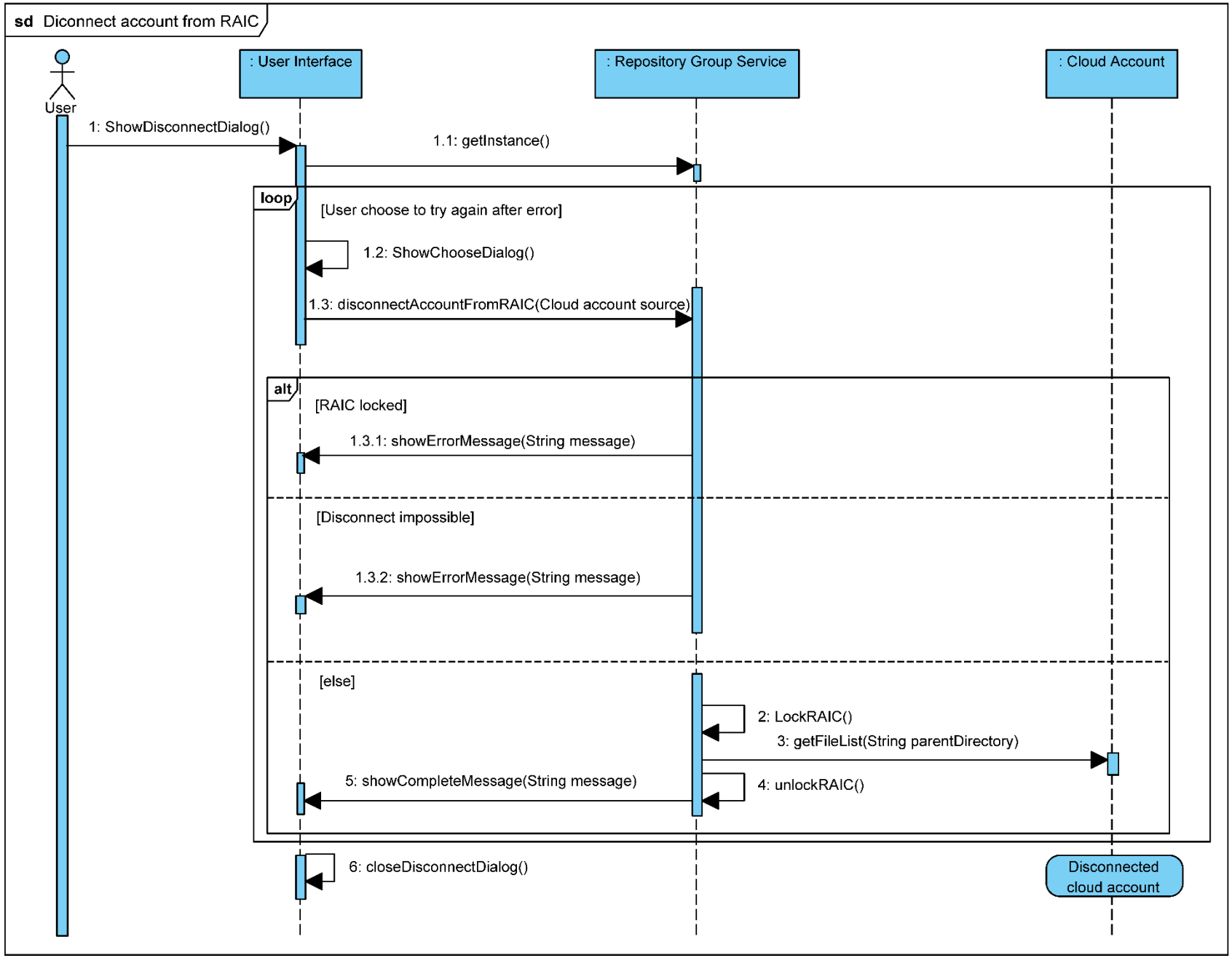


20. Диаграмма состояний пользовательского интерфейса

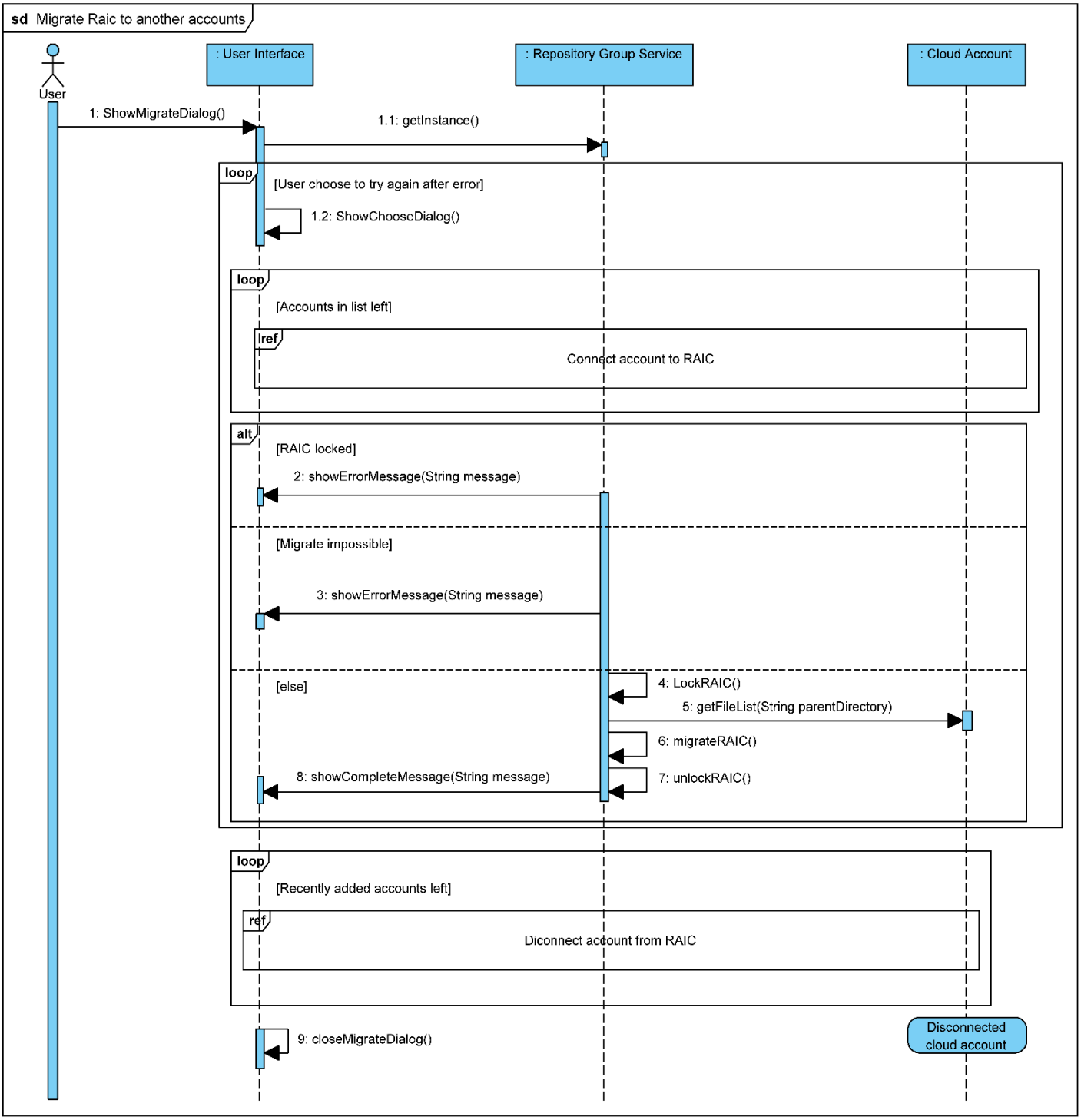
Диаграммы последовательности описывают логику работы приложения, сильно схожую с представленными ранее диаграммами деятельности, но с точки зрения взаимодействия программных компонентов, поэтому каких-либо пояснений к ним делать не нужно. Как уже говорилось ранее, не предусматривалось формирование четкого алгоритма распределения данных между облачными аккаунтами, поэтому с этой точки зрения диаграммы последовательности могут показаться неполными.



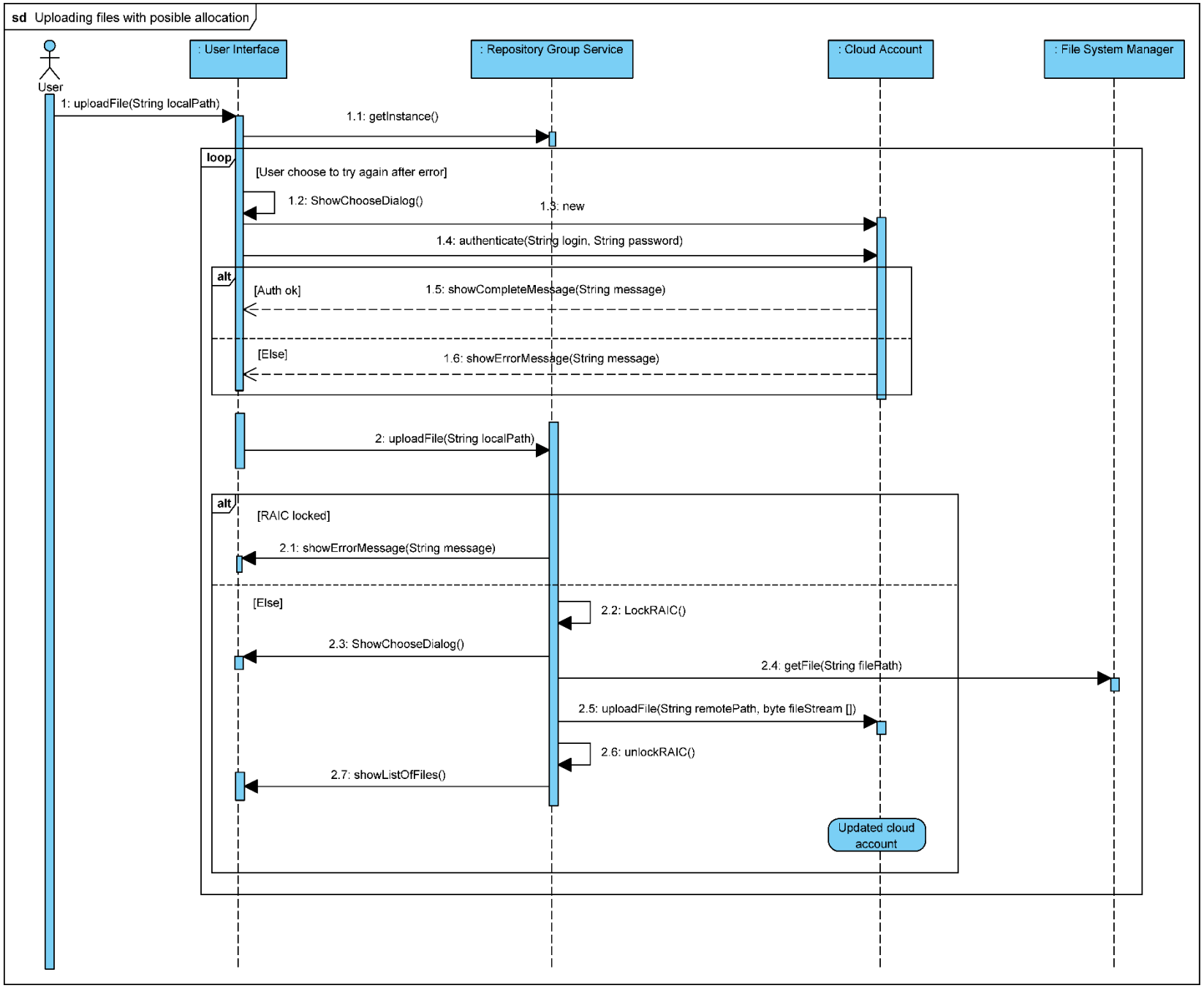
21. Диаграмма подключения аккаунта к хранилищу



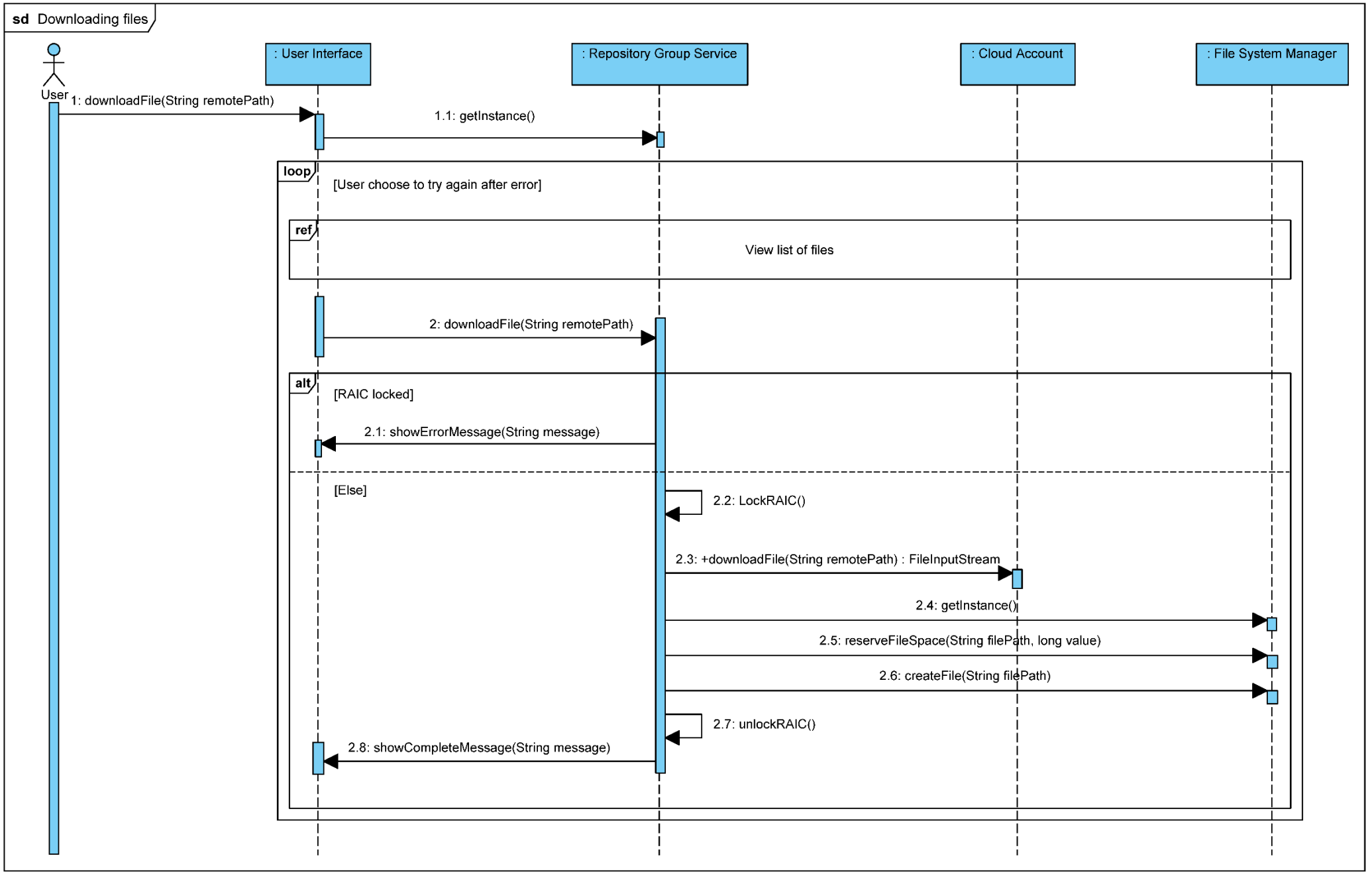
22. Диаграмма отключения аккаунта от системы



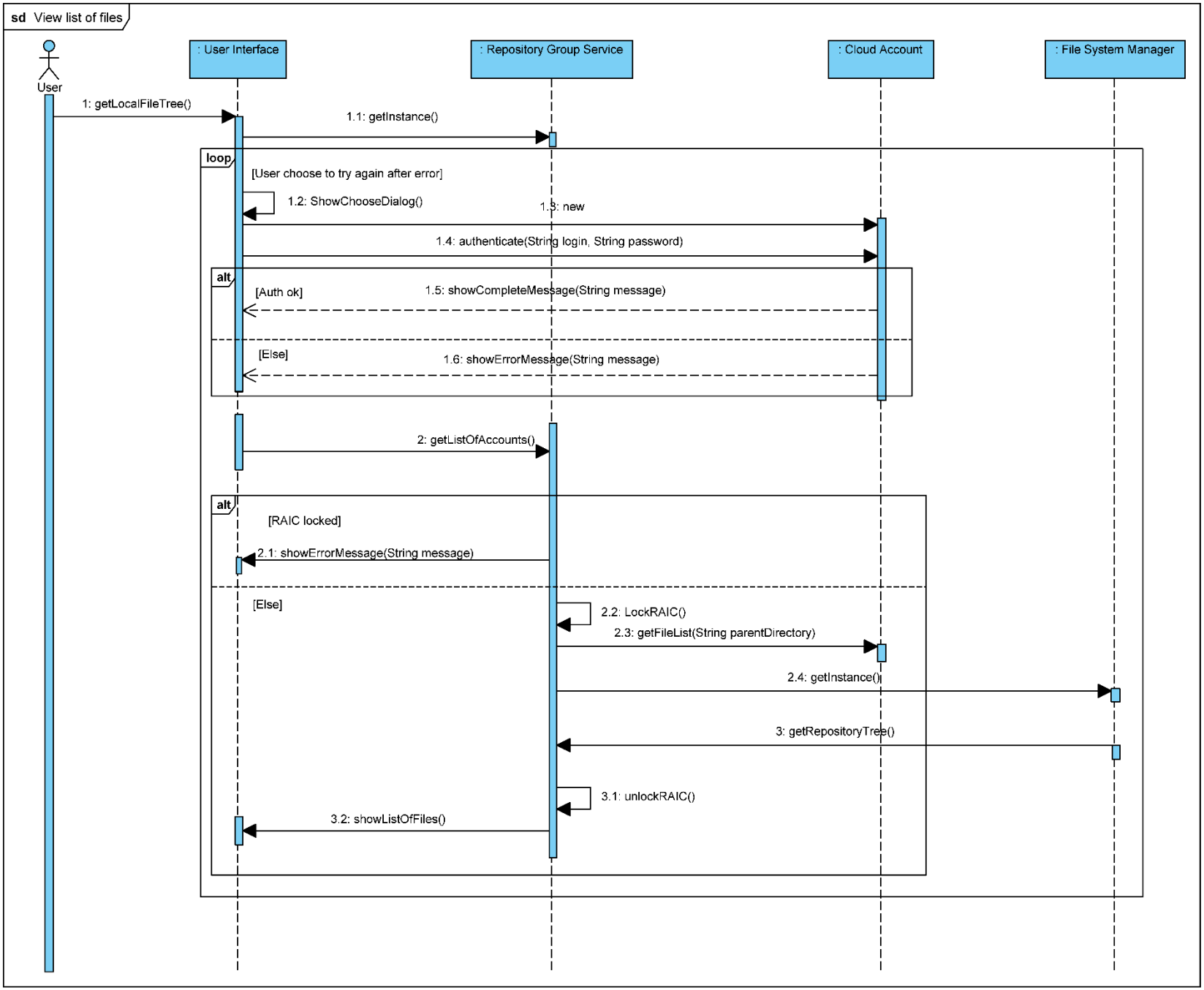
23. Диаграмма переноса данных с одних аккаунтов на другие



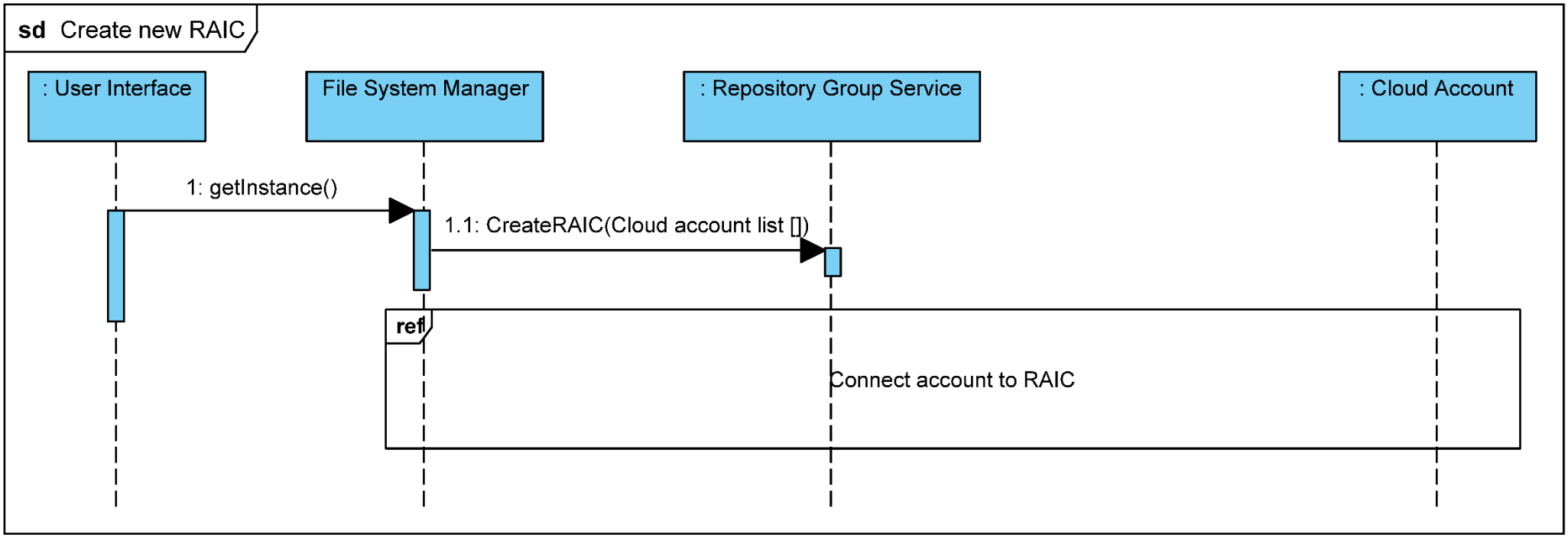
24. Диаграмма закачивания файлов при условии, что это возможно



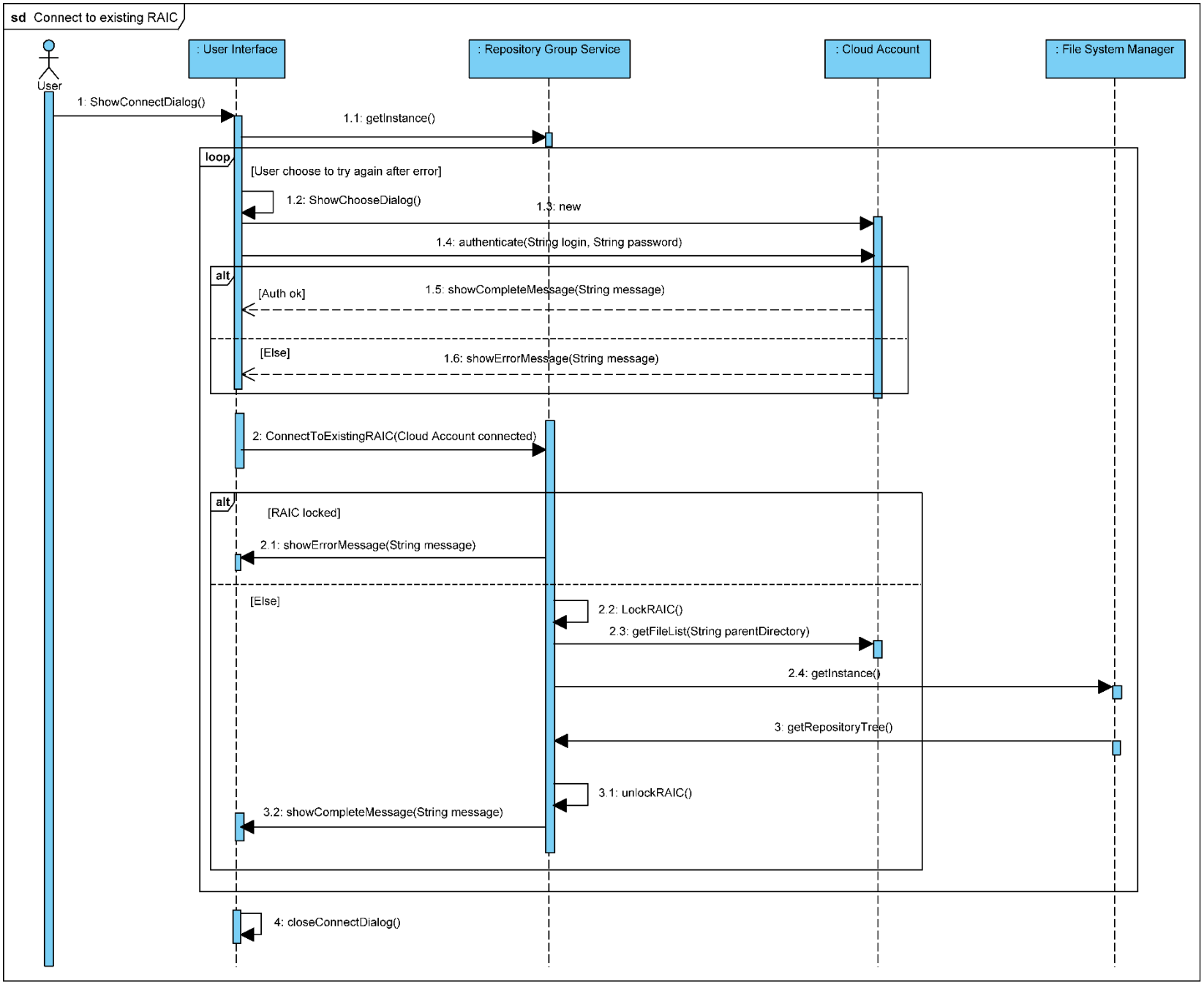
25. Диаграмма скачивания файлов



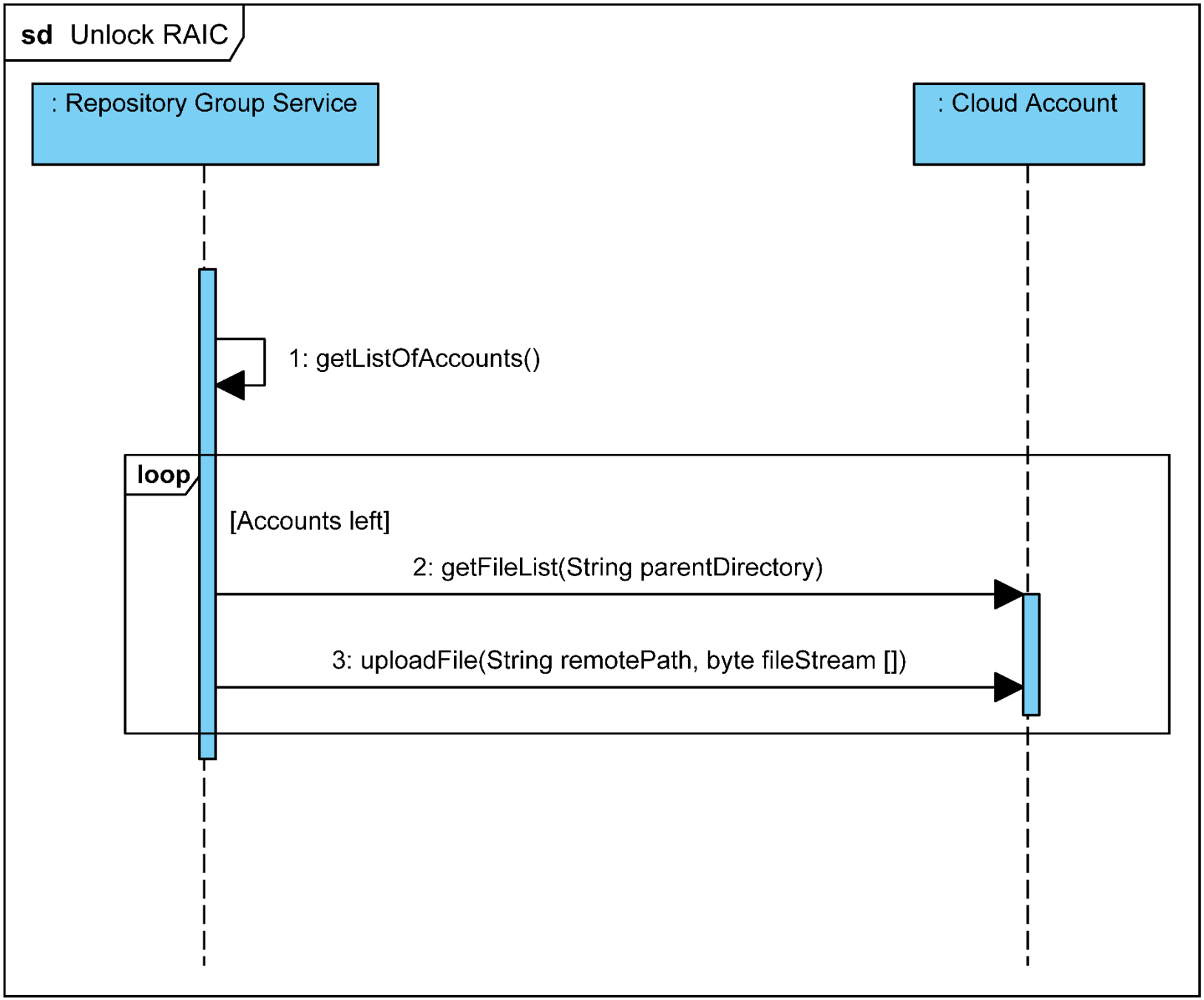
26. Диаграмма просмотра списка файлов



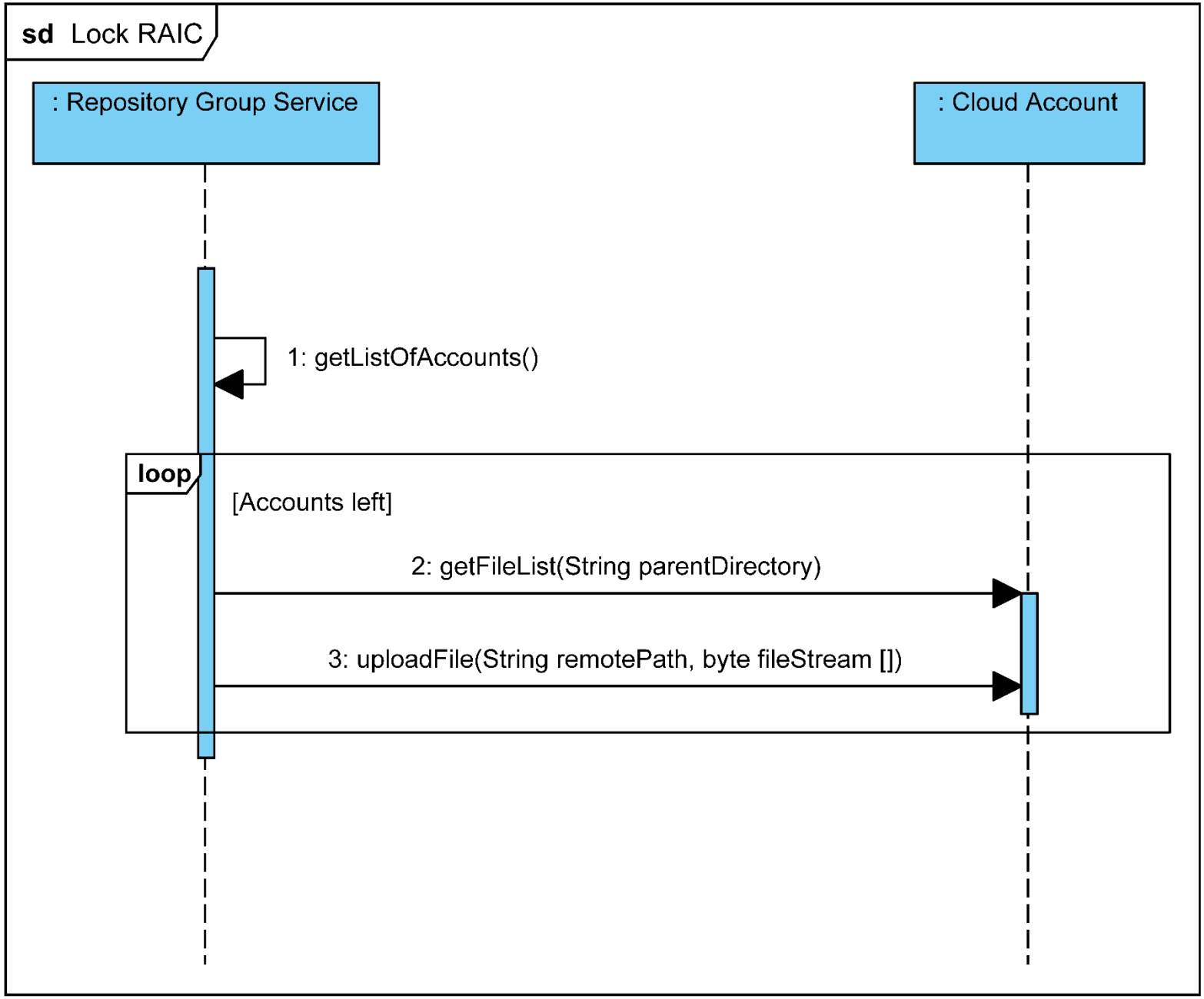
27. Диаграмма создания нового хранилища



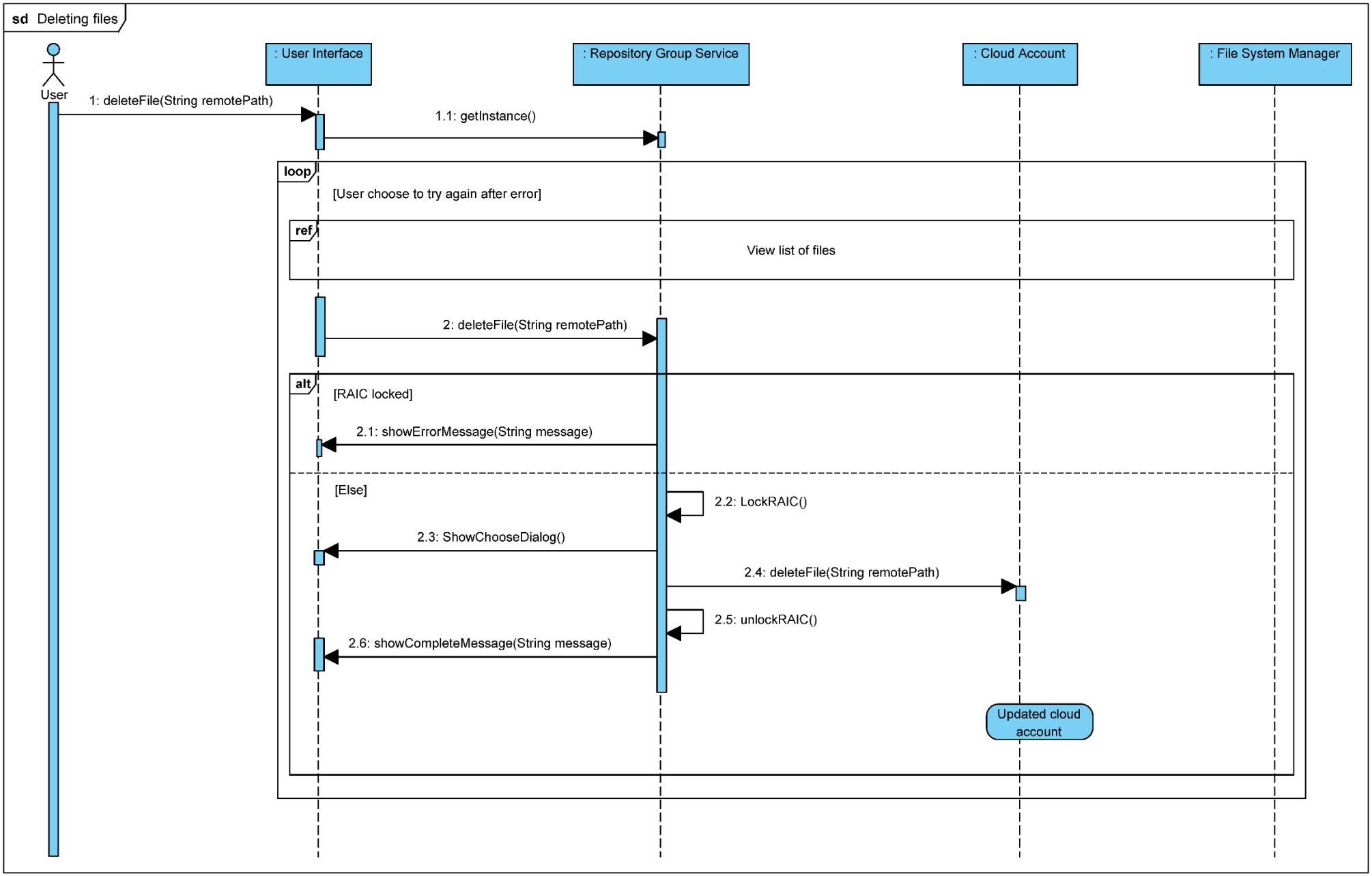
28. Диаграмма подключения к существующему хранилищу



29. Диаграмма разблокировки хранилища



30. Диаграмма блокировки хранилища



31. Диаграмма удаления файлов