Атомарное действие — это INSERT UPDATE(I,U) или DELETE(D) для одной таблицы, от одного сервера. Атомарное действие соответствует каждому реальному изменению данных в системе 1С.

От каждого сервера идет поток атомарных действий. При этом в последовательности атомарных действий мы можем безболезненно менять местами только соседние INSERT и UPDATE.

Реальная обработка атомарных действий для каждого сервера 1С происходит с некоторым периодом(10-30 минут)

Предложены две схемы:
-с использованием нумерованных пакетов с действиями
-напрямую с использованием потока атомарных действий

Проблема 1 — возможная избыточность.
В случае если в момент времени t1 было произведено атомарное действие INSERT для таблицы T, затем через минуту был произведен DELETE для тех же данных, то для временного интервала 2 минуты или больше нет смысла писать оба этих изменения. В случае, если мы используем пакеты, то данная проблема может быть решена, если же мы напрямую используем поток атомарных изменений, то оба этих изменения будут произведены также и на транспортном портале.

Т.к. нагрузка на сервер не будет высокой, то думаю, что проблему возможной избыточности можно считать не актуальной.

Проблема 2 — упорядоченность:
Атомарные действия должны быть упорядоченны относительно D и (I,U).
Действительно, если сначала в 1С был произведен INSERT, а затем DELETE, то данных нет. Но во время выполнения этих атомарных изменений на транспортном портале если сначала выполнится DELETE, а потом INSERT, то возникнет несогласованность данных. Таким образом если использовать пакеты, то нужно либо сами пакеты разделять на те, что D или (I,U). Либо внутри пакета, сами действия(необязательно атомарные) должны идти в строгом порядке один за другим.
В случае, если мы используем поток атомарных действий, то мы просто должны их дописывать каждый раз в конец файла. А те атомарные действия, подтверждения по которым пришли от транспортного портала, удалять из начала файла.

В целом думаю, что никаких принципиальных проблем со схемой потока атомарных действий нет. Так должно быть проще.

Минусы в таком подходе — это проблема 1, и увеличенный размер файла обмена.
Из плюсов это простота реализации.

Итого имеем следующую схему:
На сервере транспортного портала есть каталог в котором для каждого сервера 1С есть по два файла, например MOS.IN и MOS.OUT.

1. В файл MOS.IN 1С записывает все атомарные действия которые были произведены и не были записаны в транспортный портал. Атомарные действия должны быть упорядоченны по времени возникновения. При этом у каждого атомарного действия должен генерироваться ключ(ID).

2. Транспортный портал забирает эти данные и записывает в файл MOS.OUT последний успешно выполненный ID(все не нужно т. к. данные упорядочены).
3. 1С ждет 10 минут.
4. 1С считывает последний успешно выполненный ID из MOS.OUT.
5. 1С удаляет все атомарные действия из MOS.IN которые шли до ID, и дописывает в конец файла новые атомарные действия.
6. Переход к шагу (2)

Новый формат файла MOS.IN

```
{
  "dataFrom1C": {
    "server": "MOS",
    // время создания файла
    "created": "2015-11-20 12:05:30 GMT+0",
    "actions": [
      // данная запись означает, что в системе 1С появилось два новых пункта
      {
        "ID": "e4fgh84fg845gf4",
        "insertPoints": [
          {
            "pointName": "склад на алексеевской",
            "region": "московская область",
            "city": "москва",
            "index": "141600",
            "email": "bla@bla.com",
            "phoneNumber": "+7901567892",
            "pointType": "WAREHOUSE"
          },
          {
            "pointName": "торговый представитель в твери",
            "region": "тверская область",
            "city": "тверь",
            "index": "123456",
            "email": "bla1@bla.com",
            "phoneNumber": "+7904567892",
            "pointType": "AGENCY"
          }
        ]
      },
      // данная запись означает, из БД 1С был удален один пункт.
      {
        "ID": "greig5ig5i685689g",
        "deletePoints": [
          {
            "pointName": "склад10"
          }
        ]
      },
      // далее по образцу тажке идут данные в виде массивов для других сущностей
      {
        "ID": "relnrti5gun5ign",
        "insertUsers": [
          {
            "userID": "1234",
            "lastName": "ivanov",
            ...
          }
        ]
      }
    ]
  }
}
```