28.02.2015

Universal Computational Cluster

Overall system activities, communication protocol and plugin design

Universal Computational Cluster (UCC) is meant to be used for computing NP-hard problems by distributive exact algorithms. This document presents overall system activities on UML Activity diagrams, specifies the communication protocol by presenting the Sequence Diagrams and giving the XML Schema files for all the types of messages used by the system and as an appendix gives the crucial abstract class libraries for the Task Solvers.

1. Configuration

For the whole solution to work the following parameters should be configured in a settings file (with possible override with command line parameters):

- Communications Server: listening port number
- Communications Server: backup or primary mode
- Communications Server: communications timeout (status refresh interval)
- Other components (including CS in backup mode): address and port of the CS

The CS accepts command line parameters in the following form:

```
[-port [port number]] [-backup] [-t [time in seconds]]
```

The client components accept command line parameters in the following form:

```
[-address [IPv4 address or IPv6 address or host name]] [-port
[port number]]
```

2. System activities

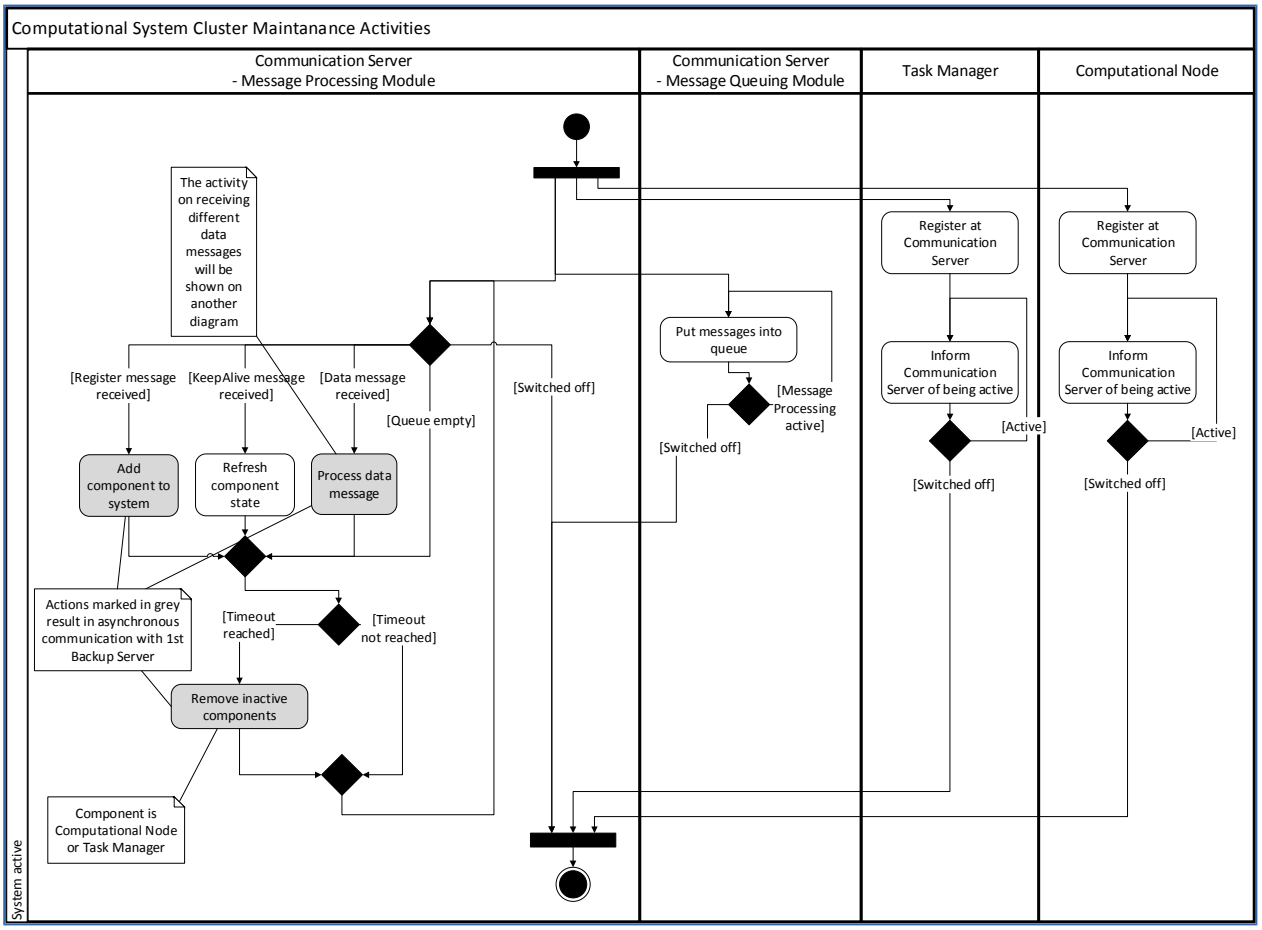

Figure 1: Overall system activities concerning the setup of the cluster

Fig. 1 presents how the components connect with each other and what activities are maintained both in the idle and active state of the cluster. Task Manager (TM) and Computational Node (CN) register to the Communication Server (CS) and constantly inform CS of their state. The TCP/IP connection is opened for each Register, KeepAlive and Data message and closed when the Communication Server sends all messages queued and possible to accept by the node that sent the message.

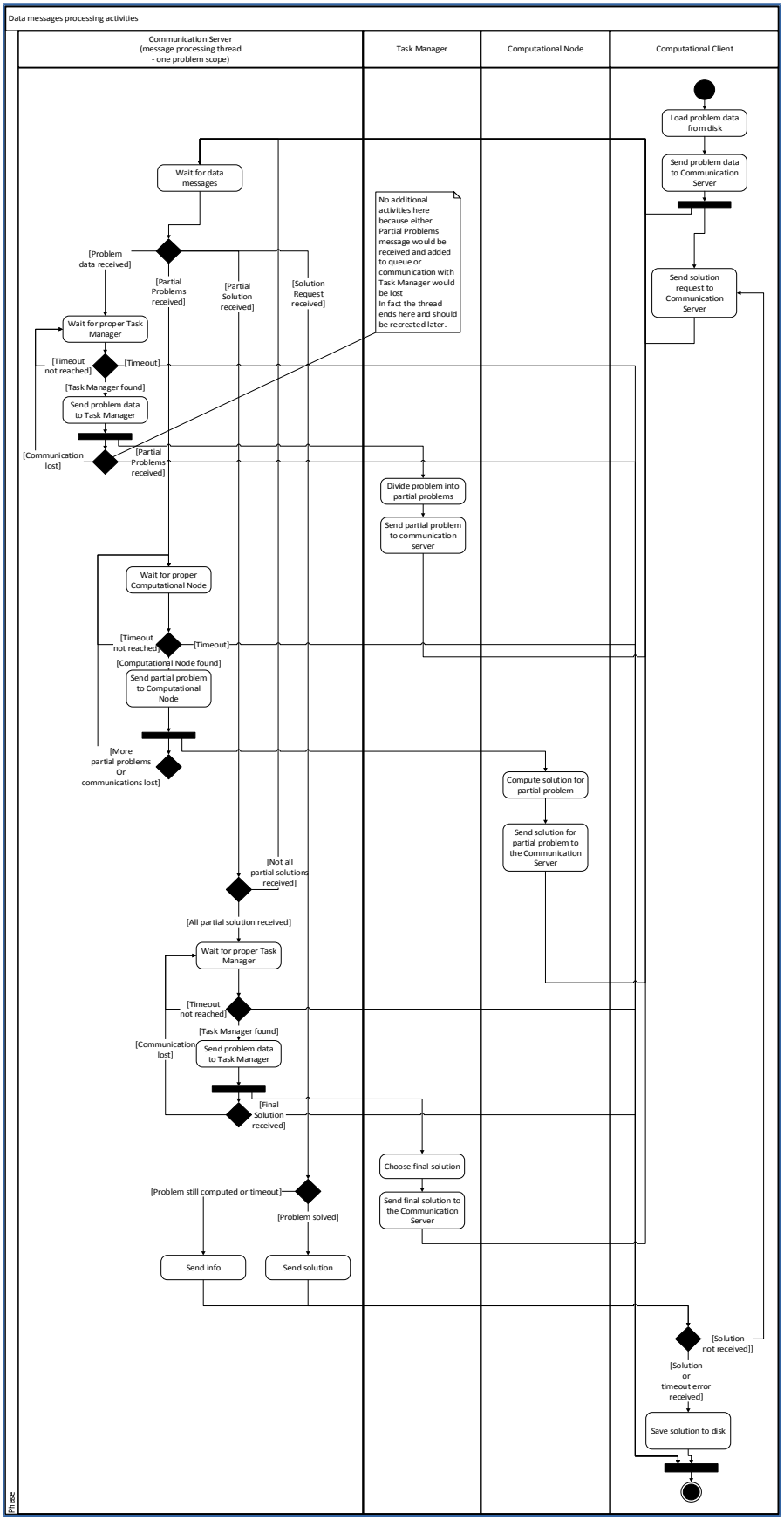

Figure 2: Single task computation activity of the cluster

Fig. 2 shows all the additional activities of the system after Computational Client (CC) connects to it and requests computations for a given problem to be performed. Both figures represent only one instance of each of the CN, TM and CC but possibly there could (and should) be many of them.

The important thing is that CS is always awaiting new messages (which are read almost instantly), regardless of the state of various problems send for computation. The processing of each of the message could be done separately and in parallel by different threads and the current state of problems is stored in the memory and not as a state of the system.

Fig. 3 shows the activities performed by the chain of Communication Server. The Backup CS form a sort of chain with each Communication Server synchronizing its data with the next. The important information that are synchronized are the existing CN and TM and their current activities and the data of tasks, partial problems, partial solutions and final solutions. Each backup server connects first with the main CS but if it is not the first backup server it registers with the current last backup server. In that way, each backup server needs to store only one set of data to be synchronized. The current information is sent to the backup CS when it registers with the CS of a higher level and is updated after each

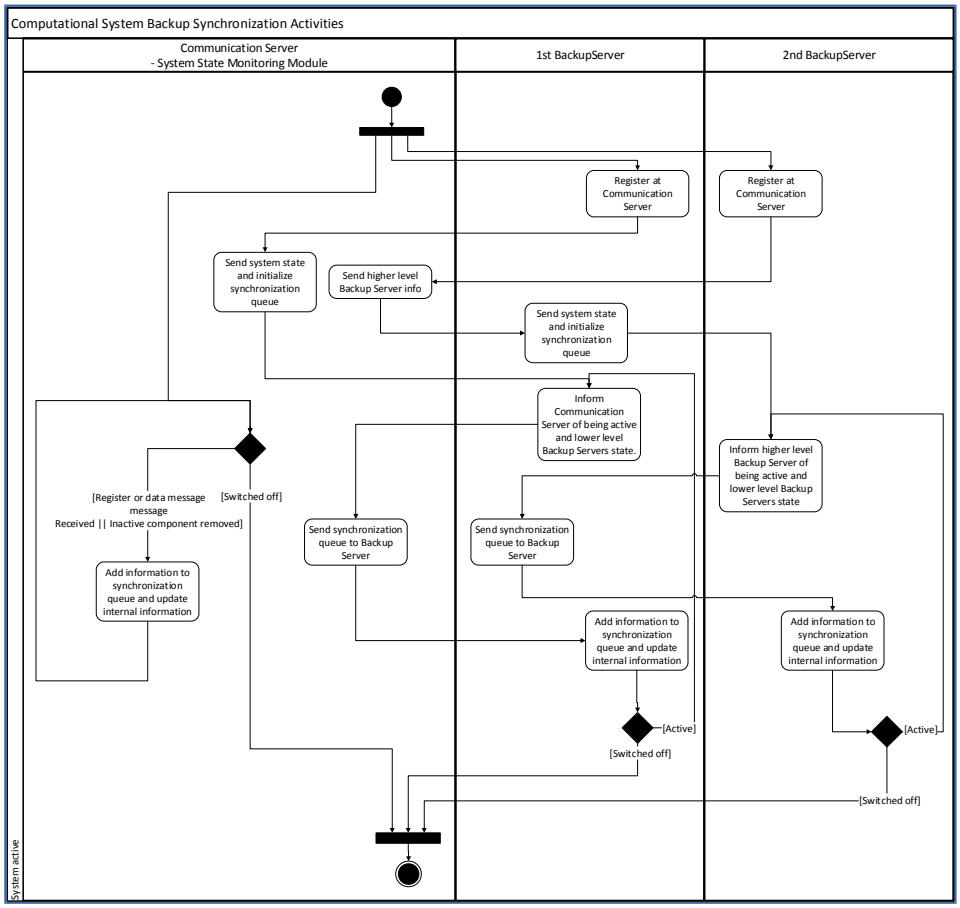

Figure 3: System activities for synchronizing data on Communication Servers

3. Communication protocol design

The system uses the TCP/IP protocol in order to send messages. Only the CS has a listens on a given port and all the other components start with connecting to CS. All the messages are a text XML messages.

The communication is maintained in a passive way: the client components connect to the CS, send their messages, wait for response messages (possibly many) and then the connection is closed by the CS. In that way only the CS' addresses and ports should be known to all of the components which allows for easier distribution of computations using the Internet and not only LAN. If messages are sent during a single connection they are separated with the sign with decimal code equal 23 (ETB - End transmission blocks).

Fig. 4 and 5 presents scenario with two TM, two CN, one CC and one backup CS. The scenario shows that each of the TM could be chosen for dividing the problem/merging the solutions and presents what happened when one of the CN crashes and then the CS crashes. Fig 4. presents the sequence

of messages passed for registration and maintenance while fig. 5 focuses on the messages passed while the computation is ongoing (the looped message exchange presented on fig. 4 is still active).

Register message

Register message is sent by TM, CN and Backup CS to the CS after they are activated. In the register message they send their type `TaskManager`, `ComputationalNode` or `CommunicationServer`; the type of problems they could solve (if applicable) and the computational power of the component (note that the protocol would support both the registration of many components from the same computer with one thread and registration of one component with many threads). When the register message is used to inform Backup CS, the Id field should be non-empty and if used to delete component from the CS or Backup CS, Deregister should be set to true.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name="Register">
    <xs:complexType>
      <xs:sequence>
        <!-- defines the type of node (either TM, CN or CS) -->
        <xs:element name="Type">
          <xs:simpleType>
            <xs:restriction base="xs:string">
              <xs:enumeration value="TaskManager" />
              <xs:enumeration value="ComputationalNode" />
              <xs:enumeration value="CommunicationServer" />
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
       <!-- gives the list of names of the problems which could be solved (probably sth
       like DVRP-[group no.]) -->
        <xs:element name="SolvableProblems">
          <xs:complexType>
            <xs:sequence>
              <xs:element  minOccurs="0" maxOccurs="unbounded" name="ProblemName"
                     type="xs:string" />
            </xs:sequence>
          </xs:complexType>
        </xs:element>
       <!-- the number of threads that could be efficiently run in parallel -->
       <xs:element name="ParallelThreads" type="xs:unsignedByte" />
       <!-- OPTIONAL when used to inform Backup Server of the need to remove element
       should be set to true -->
       <xs:element name="Deregister" type="xs:boolean" minOccurs="0" />
       <!-- OPTIONAL when used to inform Backup Server of the need
       to add/remove element should be set to ID given by main server →
       <xs:element name="Id" type="xs:unsignedLong" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Register Response message

Register Response message is sent as a response to the Register message giving back the component its unique ID and informing how often it should sent the Status message.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="RegisterResponse">
    <xs:complexType>
      <xs:sequence>
       <!-- the ID assigned by the Communication Server -->
       <xs:element name="Id" type="xs:unsignedLong" />
       <!-- the communication timeout configured on Communication Server -->
       <xs:element name="Timeout" type="xs:unsignedInt" />
       <xs:element name="BackupCommunicationServers">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="BackupCommunicationServer" minOccurs="0">
                <xs:complexType>
                  <xs:attribute name="address" type="xs:anyURI" />
                  <xs:attribute name="port" type="xs:unsignedShort" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

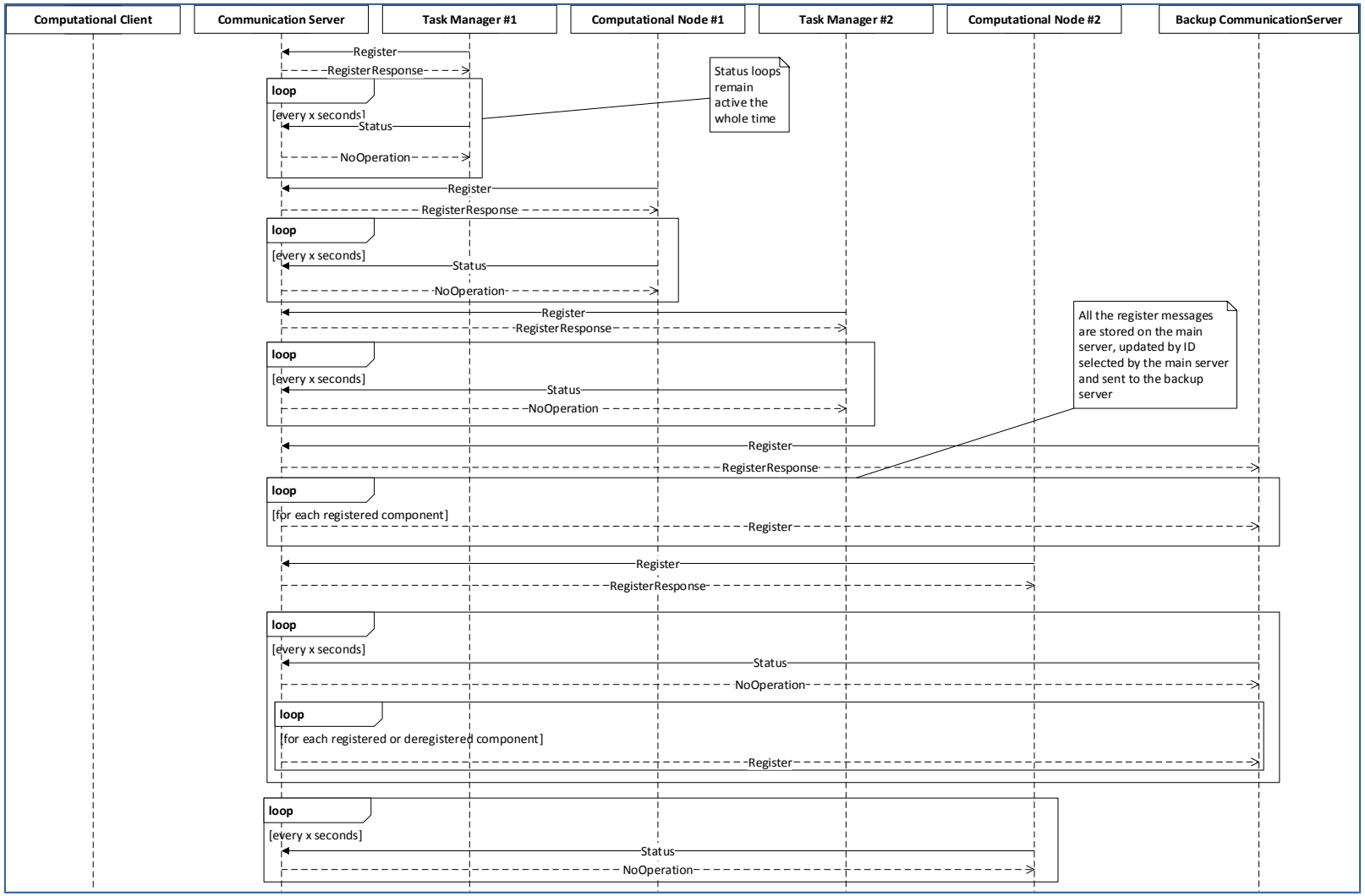

Figure 4: Sequence diagram for registration and maintenance messages

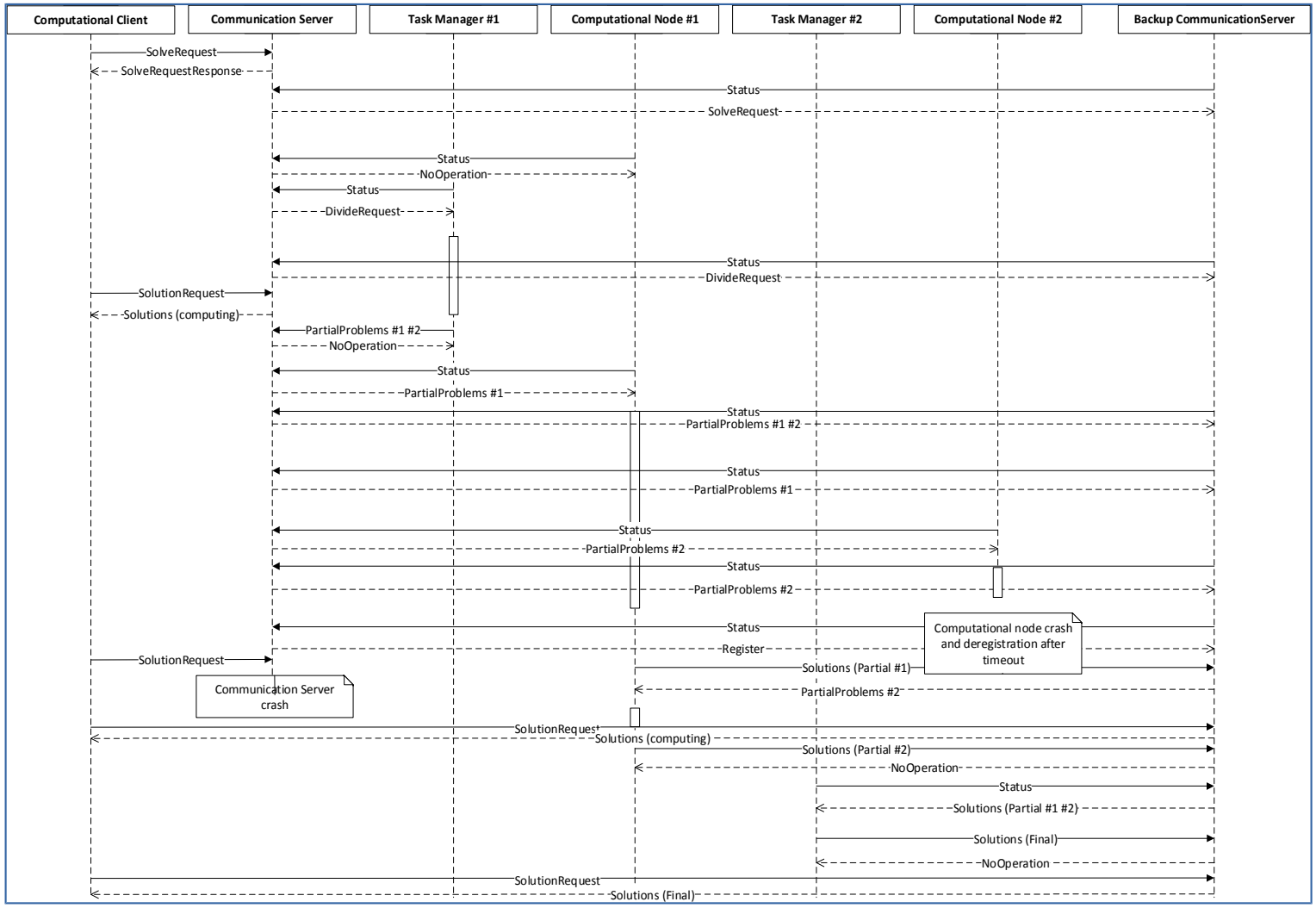

Figure 5: Sequence diagram for data messages with Computational Node and Communication Server failures

Status message

Status message is sent by TM, CN and Backup CS to CS at least as frequent as a timeout given in Register Response. In the Status message the component reports the state of its threads, what they are computing (unique problem instance id, task id within the given problem instance, the type of problem instance) and for how long.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace=http://www.mini.pw.edu.pl/ucc/
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Status">
    <xs:complexType>
      <xs:sequence>
       <!-- the ID of node (the one assigned by server) -->
       <xs:element name="Id" type="xs:unsignedLong" />
       <!-- list of statuses for different threads -->
       <xs:element name="Threads">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="Thread">
                <xs:complexType>
                  <xs:sequence>
                     <!-- information if the tread is currently computing something  -->
                    <xs:element name="State">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:enumeration value="Idle" />
                          <xs:enumeration value="Busy" />
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:element>
                     <!-- how long it is in given state (in ms) -->
                     <xs:element name="HowLong” type="xs:unsignedLong" minOccurs="0"
                          />
                     <!-- the ID of the problem assigned when client connected -->
                     <xs:element minOccurs="0" name="ProblemInstanceId"
                     type="xs:unsignedLong" />
                     <!-- the ID of the task within given problem instance -->
                    <xs:element minOccurs="0" name="TaskId" type="xs:unsignedLong" />
                     <!-- the name of the type as given by TaskSolver -->
                    <xs:element minOccurs="0" name="ProblemType" type="xs:string" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

No Operation message

No Operation message is sent by the CS in response to Status messages. It is used to inform the components about the current list of backup servers. It could be send in conjunction with other messages.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="NoOperation">
    <xs:complexType>
      <xs:sequence>
        <!-- The list of backup servers -->
        <xs:element name="BackupCommunicationServers">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="BackupCommunicationServer" minOccurs="0">
                <xs:complexType>
                  <!-- server address -->
                  <xs:attribute name="address" type="xs:anyURI" />
                  <!-- server port -->
                  <xs:attribute name="port" type="xs:unsignedShort" />
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Solve Request message

Solve Request message is sent by the CC to CS. It gives the type of the problem instance to be solved, optionally the max time that could be used for computations and the problem data in base64. The same message is used to relay information for synchronizing info with Backup CS.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace=”http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SolveRequest">
    <xs:complexType>
      <xs:sequence>
       <!-- the name of the type as given by TaskSolver -->
        <xs:element name="ProblemType" type="xs:string" />
       <!-- the optional time restriction for solving the problem (in ms) -->
        <xs:element minOccurs="0" name="SolvingTimeout" type="xs:unsignedLong" />
       <!-- the serialized problem data -->
       <xs:element name="Data" type="xs:base64Binary" />
       <!-- the ID of the problem instance assigned by the server -->
        <xs:element name="Id" type="xs:unsignedLong" minOccurs="0" />
```

```
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Solve Request Response message

Solve Request Response message is sent by CS to CC as an answer for the Solve Request. It provides CC with unique identifier of the problem instance.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace=”http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SolveRequestResponse">
    <xs:complexType>
      <xs:sequence>
       <!-- the ID of the problem instance assigned by the server -->
       <xs:element name="Id" type="xs:unsignedLong" minOccurs="0" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Divide Problem message

Divide Problem is sent to TM to start the action of dividing the problem instance to smaller tasks. TM is provided with information about the computational power of the cluster in terms of total number of available threads. The same message is used to relay information for synchronizing info with Backup CS.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace=”http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="DivideProblem">
    <xs:complexType>
      <xs:sequence>
        <!-- the problem type name as given by TaskSolver and Client -->
        <xs:element name="ProblemType" type="xs:string" />
        <!-- the ID of the problem instance assigned by the server -->
        <xs:element name="Id" type="xs:unsignedLong" />
        <!-- the problem data -->
        <xs:element name="Data" type="xs:base64Binary" />
        <!-- the total number of currently available threads  -->
        <xs:element name="ComputationalNodes" type="xs:unsignedLong" />
        <!-- the ID of the TM that is dividing the problem -->
        <xs:element name="NodeID" type="xs:unsignedLong" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Solution Request message

Solution Request message is sent from the CC in order to check whether the cluster has successfully computed the solution. It allows CC to be shut down and disconnected from server during computations.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace=”http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SolutionRequest">
    <xs:complexType>
      <xs:sequence>
        <!-- the ID of the problem instance assigned by the server -->
        <xs:element name="Id" type="xs:unsignedLong" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Partial Problems message

Partial problems message is sent by the TM after dividing the problem into smaller partial problems. The data in it consists of two parts – common for all partial problems and specific for the given task. The same Partial Problems schema is used for the messages sent to be computed by the CN and to relay information for synchronizing info with Backup CS.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace=”http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="SolvePartialProblems">
    <xs:complexType>
      <xs:sequence>
        <!-- the problem type name as given by TaskSolver and Client -->
        <xs:element name="ProblemType" type="xs:string" />
        <!-- the ID of the problem instance assigned by the server -->
        <xs:element name="Id" type="xs:unsignedLong" />
        <!-- the data to be sent to all Computational Nodes -->
        <xs:element name="CommonData" type="xs:base64Binary" />
        <!-- optional time limit – set by Client (in ms) -->
       <xs:element minOccurs="0" name="SolvingTimeout" type="xs:unsignedLong" />
       <xs:element name="PartialProblems">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="PartialProblem">
                <xs:complexType>
                  <xs:sequence>
                    <!-- Id of subproblem given by TaskManager -->
                    <xs:element name="TaskId" type="xs:unsignedLong" />
                    <!-- Data specific for the given subproblem -->
                    <xs:element name="Data" type="xs:base64Binary" />
                    <!-- the ID of the TM that is dividing the problem -->
                    <xs:element name="NodeID" type="xs:unsignedLong" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Solutions message

Solutions message is used for sending info about ongoing computations, partial and final solutions from CN, to TM and to CC and to relay information for synchronizing info with Backup CS. In addition to sending task and solution data it also gives information about the time it took to compute the solution and whether computations were stopped due to timeout.

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://www.mini.pw.edu.pl/ucc/"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Solutions">
    <xs:complexType>
      <xs:sequence>
        <!-- the problem type name as given by TaskSolver and Client -->
        <xs:element name="ProblemType" type="xs:string" />
        <!-- the ID of the problem instance assigned by the server -->
        <xs:element name="Id" type="xs:unsignedLong" />
        <!-- common data which was previously sent to all Computational Nodes (possibly
could be stored on server as TaskManagers could have changed) -->
        <xs:element minOccurs="0" name="CommonData" type="xs:base64Binary" />
        <xs:element name="Solutions">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" name="Solution">
                <xs:complexType>
                  <xs:sequence>
                    <!-- Id of subproblem given by TaskManager – no TaskId for
              final/merged solution -->
                    <xs:element minOccurs="0" name="TaskId" type="xs:unsignedLong" />
                    <!-- Indicator that the computations ended because of timeout -->
                    <xs:element name="TimeoutOccured" type="xs:boolean" />
                    <!-- Information about the status of result (Partial/Final) or status
              of computations (Ongoing) -->
                    <xs:element name="Type">
                      <xs:simpleType>
                        <xs:restriction base="xs:string">
                          <xs:enumeration value="Ongoing" />
                          <xs:enumeration value="Partial" />
                          <xs:enumeration value="Final" />
                        </xs:restriction>
                      </xs:simpleType>
                    </xs:element>
                    <!-- Total amount of time used by all threads in system for computing
              the solution / during the ongoing computations (in ms) -->
                    <xs:element name="ComputationsTime" type="xs:unsignedLong" />
                    <!-- Solution data -->
                    <xs:element name="Data" minOccurs="0" type="xs:base64Binary" />
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

4. Q&A (in polish)

W dniu 27 lutego 2014 09:43 użytkownik Michal Cwienczek <cwienczekm@student.mini.pw.edu.pl> napisał:

>Czy moglibysmy prosic o bardziej szczegolowy opis pol w wiadomosiach XML?

Prośba uwzględniona.
Do 10.03. dokumentacja zostanie uzupełniona w tym zakresie.

W dniu 27 lutego 2014 09:44 użytkownik Michal Cwienczek <cwienczekm@student.mini.pw.edu.pl> napisał:

>Czy beda z gory zadane TaskSolvery i czy dostaniemy je do "podpiecia" na zywo do projektu czy mamy
>sami implementowac przed zlozeniem projektu?

Implementacja Task Solverów jest jednym z głównych zadań projektów,
natomiast wszystkie Task Solvery będą miały za zadanie znajdować dokładne rozwiązania dla instancji problemu DVRP i jeżeli będą Państwo wytwarzali w technologii .NET to Task Solver ma być zrealizowany w formie wtyczki implementującej dostarczoną klasę abstrakcyjną - w razie potrzeby będziemy się starali dostarczać analogiczne biblioteki do innych technologii.

Więcej (oprócz przesyłanych w poprzednim semestrze) publikacji i materiałów nt. DVRP znajdą Państwo m.in. u mnie na stronie przedmiotu MSI2.

W dniu 27 lutego 2014 09:47 użytkownik Michael Cwienczek <michael.cwienczek@gmail.com> napisał:

>Chcielibysmy zaproponowac zobostrzenie (zawężenie) aby komunikacja odbywala sie po protokole
>HTTP zgodnie z jego standardami. Swoja propozycje usprawiedliwiamy tym, ze jest to juz ogolny
>standard komunikacji na swiecie.

Nie zgadzam się - dodatkowy narzut związany z protokołem HTTP (zarówno po stronie implementacyjnej jak i komunikacyjnej) jest tutaj całkowicie zbędny.

W dniu 27 lutego 2014 09:54 użytkownik Michal Cwienczek <cwienczekm@student.mini.pw.edu.pl> napisał:

>wydaje mi sie ze brak jest w dokumentacji opisu jak CN czy tez CC odkrywa obecnosc CS w systemie. Czy
>jest wysylany jakis broadcast? Jak serwer informuje inne wezly o tym jaki jest jego IP?

Serwer o niczym nie informuje - jego IP i port są traktowane jako znane wartości i stanowią element konfiguracji systemu (sekcja Configuration) w dokumentacji.

W dniu 1 marca 2014 21:53 użytkownik Konrad Oleksiuk <oleksiukk@student.mini.pw.edu.pl> napisał:

>Na początku podane było przez państwa, że projekt może być napisany w dowolnej technologii. Uznaję >więc, że to nie tylko C# i Java, ale też Ruby, Python, Erlang czy cokolwiek.

Oczywiście, proszę się nie sugerować dostarczoną DLLką, powstała ona ponieważ znakomita większość studentów PL, będzie pisać w .NET.

>1 .Czy jest jakiś konkretny powód dla którego jako schemat komunikacji został wybrany wspierany >głównie przez MS xml + schema? Praktycznie żaden język nie ma tak dobrego wsparcia jak C#, co >teoretycznie obliguje do korzystania z niego. Czy nie moglibyśmy wprowadzić bardziej generycznego >formatu, jakim jest JSON? Rozmawiałem z częścią osób i również podzielają moje zdanie, że podany >format komunikacji jest praktycznie nie do zaakceptowania przy innym języku, a implementacja obsługi >tego formatu mija się z celem, bo nikt nie zamierza poświęcać po 15-30h tygodniowo, żeby to w ogóle >działało.

XML Schema jest:

a) jednym z powszechnie przyjętych sposobów definiowania komunikatów
b) co najmniej równie wspierany jest przez Javę (JAXB), Pythona (PyXB) jak i pewnie pozostałe technologie...
Zwykle Google: [technologia] code generation from XSD pomaga znaleźć co trzeba.

>2. <member
>name="M:UCCTaskSolver.TaskSolver.Solve(System.Byte[],System.Byte[],System.TimeSpan)”>
>To również można podciągnąć pod punkt pierwszy: Wymaganie używania C# jako TaskSolver

Jak we wstępie - ten XML to tylko dokumentacja do DLLki.

>3. Ktoś jest w stanie to [XML] w ogóle wyjaśnić? Bo dokumentacja zupełnie pomija, do czego w ogóle >miałoby to służyć.

Problem był zgłaszany, do 10.03. pola zyskają opisy.

>4. The system uses the TCP/IP protocol in order to send messages. Only the CS has a listens on a given >port and all the other components start with connecting to CS. All the messages are a text XML >messages.

>Nie bardzo rozumiem, dlaczego akurat TCP/IP i konkretny port. Czy nie łatwiejszym rozwiązaniem jest >zastosowanie po prostu bardziej abstrakcyjnego protokołu http, który pozwala na standaryzację >połączenia / wysyłania danych między elementami, niezależnie od języka?

To protokół HTTP jest bardziej konkretny niż TCP/IP, HTTP korzysta z TCP/IP a więc rezygnujemy z narzutów komunikacyjnych (niewielkich) jak i komponentowych (potencjalnie dużych).
Przesyłanie XMLi/JSONów jako stringów jest również niezależne od języka.

>5. Czy projekty z grupy anglojęzycznej w ogóle były brane pod uwagę?

Ze względu na to, że państwo mieli nieco dłuższy termin na oddawanie dokumentacji nie byłem w stanie uwzględnić jej w procesie wyboru, scalania i kompilacji do wersji obowiązującej.

W dniu 2 marca 2014 22:25 użytkownik kozickit <kozickit@student.mini.pw.edu.pl> napisał:

>Czy możliwość posiadania wielu wątków dla jednego komponentu powinna dotyczyć też TaskManager-a?

Zgodnie z generalnym celem projektu - wymienionym w preambule do pierwszych wytycznych (zeszły semestr) - celem projektu jest napisanie systemu o maksymalnej wydajności, w ramach posiadanych zasobów.

Wydaje się, że wielowątkowość TM (poza ew. wątkami przyjmującymi i obsługującym wiadomości) może nie być istotna ze względu na możliwość współistnienia TM i CN na jednej maszynie, natomiast wielowątkowy TM ma znaczenie, jeżeli zakładają Państwo wykonywanie intensywnych obliczeń przez TM przy scalaniu zadań - wtedy jego wielowątkowość będzie miała znaczenie.

W dniu 2 marca 2014 22:28 użytkownik kozickit <kozickit@student.mini.pw.edu.pl> napisał:

>Zmiana nazwy wiadomości `PartialProblems` na `SolvePartialProblems`, aby uniknąć używania w >wiadomości takiej samej nazwy elementu w różnym kontekście.

Uwzględnię przy nanoszeniu poprawek.

W dniu 2 marca 2014 22:30 użytkownik kozickit <kozickit@student.mini.pw.edu.pl> napisał:

>Zmiana elementów SolvingTimeout, oraz ComputationsTime z obecnego formatu typu String na czas >wyrażony w milisekundach typu unsignedLong.

Dobry pomysł.
Zmienimy.

W dniu 2 marca 2014 22:32 użytkownik kozickit <kozickit@student.mini.pw.edu.pl> napisał:

>W wiadomości RegisterResponse element Timeout powinien być ustalony z góry, ponieważ grupy >mogłyby przyjąć zupełnie różne wartości tego elementu, powodując problemy w komunikacji >komponentów różnych grup.

Są 2 timeouty:

1. pierwszy dotyczy timeout'ów komunikacji - czyli jak często komponenty powiadamiają przy pomocy statusów, że żyją - ten timeout jest ustawiany tylko na serwerze a pozostałe komponenty otrzymują go jako parametr dopiero po skontaktowaniu się z serwerem
2. dotyczy czasu na obliczenia - ustala go CC - po tym czasie obliczenia mają się zakończyć - niekoniecznie pełnym sukcesem, ale jakimś rozwiązaniem

Także zostaje tak jak jest.

W dniu 2 marca 2014 22:37 użytkownik kozickit <kozickit@student.mini.pw.edu.pl> napisał:

>
> W obecnym kształcie biblioteka *UCCTaskSolver.dll* z abstrakcyjną klasą `TaskSolver` uniemożliwia
> poprawną implementację metody `GetInstance()`. Obecnie metoda ta w klasie pochodnej nie może
> być określona jako statyczna (co zakłada wzorzec *singleton*).
>

Dziękuję za uwagę, zdecydowanie nie przemyślałem tego fragmentu rozwiązania, zwłaszcza singletona - jeżeli już miałby być to przecież trzeba go jakoś sprytniej opakować bo task solverów może być dużo różnych.

W każdym razie ta klasa zostanie poprawiona.

W dniu 6 marca 2015 20:40 użytkownik draganl <draganl@student.mini.pw.edu.pl> napisał:
>
> Jak wygląda moment wyrejestrowania komponentu po upływie timeoutu
> (jak liczyć timeout i za pomocą czego wyrejestrować) oraz jak to się ma do
> pasywności
> serwera - kiedy informacja o wyrejestrowaniu ma trafić na backup.
>

Komponent zostaje wyrejestrowany jeżeli przestaje wysyłać wiadomości (jeżeli między kolejnymi wiadomościami będzie odstęp większy niż *timeout*). *Timeout* jest konfigurowany na serwerze i jest to czas, po którym jeżeli nie otrzymamy wiadomości od klienta (Status lub innej), uznajemy, że komponent przestał istnieć. Komponent poznaje ten czas w odpowiedzi na rejestrację.

Backup dowiaduje się o zniknięciu komponentu za pierwszym odpytaniem serwera po zniknięciu komponentu (czyli dane na serwer *backup* nie są replikowane natychmiast).

> W jakim celu w wiadomości StatusMessage jest przesyłany tag
> <xs:element name="HowLong” type="xs:unsignedLong" minOccurs="0"/>
>

W celach statystycznych - możemy w ten sposób prowadzić diagnostykę algorytmów: ile czasu liczyły i jaka była utylizacja procesorów całego systemu w trakcie ich działania.

> W jakim celu w wiadomości SolveRequest jest przesyłany tag
> <xs:element minOccurs="0" name="SolvingTimeout" type="xs:unsignedLong" />
>

To jest *timeout* dla użytkownika - można zlecić przerwanie zadania po określonym czasie - i być może otrzymać jakiś wynik.

> Na czym polega czekanie na ComputionalNode przy obsłudze timeoutu przy
> wysyłaniu w CS - czy czekamy, aż Node odpowiedni do zadania w się pojawi lub
> się zwolni, odkładając go na jakąś strukturę, którą na bieżąco sprawdzamy,
> czy też wątek obsługujący aktualnie to zadanie ma czekać wyznaczony timeout.
>

Są dwa timeouty - ustalany przez użytkownika przy definicji zadania oraz ustalany przez administratora przy stawianiu klastra. Timeout na policzenie zadania obejmuje wszystko: czekanie na węzły, czas dzielenia, czas obliczeń i czas sklejania. Po upływie timeoutu węzły powinny przestać liczyć, a serwer przechować odpowiedź (jeżeli jakaś powstała). Ten timeout może zostać potraktowany nieściśle - tzn. jeżeli niewielkim kosztem możemy uzyskać jakiekolwiek rozwiązanie problemu (np. mamy jakieś propozycje rozwiązań, ale nie scalone - to możemy je jeszcze scalić po timeoucie).

W dniu 7 marca 2015 16:36 użytkownik Kamil Sienkiewicz <sienkiewiczk@student.mini.pw.edu.pl> napisał:

> Dzień dobry, po wspólnej analizie otrzymanej dokumentacji przygotowaliśmy
> kilka pytań: 1. Czy klasa abstrakcyjna reprezentująca TaskSolver nie mogłaby
> być zastąpiona interfejsem? Umożliwiłoby to z pewnością stworzenie bardziej
> elastycznego rozwiązania?

Zakładam w przyszłości rozwój samej klasy o to, żeby coś się w niej działo, więc pozostawiam klasę abstrakcyjną.

> 2. Czy zakładamy i obsługujemy przypadek gdy
> główny CommunicationServer zostaje ponownie włączony do klastra po błędzie,
> w jaki sposób miałoby się odbyć przekazanie informacji z serweru
> backup'owego?

Jeżeli zostaje włączony to jako Backup Server po czym trzeba ubić inne serwery, żeby to on stał się głównym.

> 3. Jak klaster reaguje na wiadomości od niezarejestrowanych
> komponentów? Chodzi nam np. o sytuację, że Computational Node nie zdążył w
> odpowiednim czasie poinformować CS, że nadal "żyje" (Status message) i
> serwer go wyrejestrował, a chwilę później CN wysłał rozwiązanie swojego
> podproblemu. Podobnie sytuacja w której element przekracza czas obliczeń
> (timeout), ale go ignoruje i wysyła rozwiązanie po nim, gdy już inny element
> go oblicza ponownie (błędnie napisane elementy, informacja przydatna będzie
> do łączenia z innymi grupami).

Bardzo dobre pytanie. Czekam na propozycję schemy (analogiczną do pozostałych) i przykład(y) wiadomości błędu wraz z możliwymi komunikatami / statusami.

> 4. W jaki sposób będą sklejane wiadomości
> przesyłane od serwera do pozostałych komponentów? Będzie to jeden string
> przesłany w jednym pakiecie czy 2 stringi w oddzielnych pakietach?

Jeżeli kilka wiadomości jest wysyłanych podczas jednego połączenia, to są one oddzielane znakiem specjalnym o kodzie 23 (wg rozdziału 3.). Jeżeli chodzi o podział na pakiety: z punktu widzenia programowania będzie to czytanie i pisanie do strumienia – pakietami nie trzeba się martwić.

W dniu 7 marca 2015 17:34 użytkownik buszkom <buszkom@student.mini.pw.edu.pl> napisał:

> 1. Rozumiem że pierwsza wiadomość jaką wysyła CC do CS jest wiadomością typu
> `Solve Request` (żądanie rozwiązania zadania). Zastanawiam się czym jest
> element o nazwie `Id` w tej widomości skoro Id zadania jest wysyłane przez
> CS do CC w `Solve Request Response` czyli jako odpowiedź na `Solve
> Request`...

Jest analogicznie jak w RegisterResponse (czyli korzysta z tego tylko BackupCS), uzupełniłem w dokumencie minOccurs="0" (w pliku xsd było dobrze).

> 2. Czy CC cały czas utrzymuje połączenie z CS? Czy wiadomość
> `SolutionRequest` jest wysyłana manualnie przez użytkownika czy CC wysyła ją
> automatycznie np. co 5 sekund?

Żadne połączenia nie są utrzymywane.
SolutionRequest może być wysyłane z dowolną częstotliwością, implementacja tej częstotliwości i manualnie vs. ręcznie nie jest ściśle określona.

> 3. Dla przykładu jeśli TM rejestruje się wysyłając do CS `Register` message
> i CS odsyła mu `Reqister Response` message to następnie połączenie między
> między TM a CS jest zrywane, a w późniejszych chwilach CS identyfikuje

> powyższy TM poprzez jego ID? Czy połączenie między CS a TM i CN musi być
> cały czas utrzymywane?

Połączenie jest zrywane. Komponenty przedstawiają się swoim ID.

> 4. Mając uruchomione dwa Backup CS po awarii Primary CS ską wiadomo który
> Backup CS przejmie pracę i skąd TM, CN i CC wiedzą jaki jest nowy CS?
>

BackupCS, który jest wyżej na liście w wiadomości *NoOperation*, będący też wyżej w łańcuchu *BackupServerów,* ponieważ:

1. pierwszy się zgłosił
2. jego stan jest bliższy stanu serwera
3. nie zaburzamy łańcucha komunikacji między CSami.