Software Environment for Market Balancing Mechanisms Development, and Its Application to Solving More General Problems in Parallel Way

Mariusz Kamola

[1] Institute of Control and Computation Engineering, Warsaw University of Technology
[2] NASK (Research and Academic Computer Network)
Mariusz.Kamola@nask.pl

Abstract. A new software that supports research on algorithms for resource allocation in multi-commodity markets is presented. Thanks to a much more general data model used, and possibility of plugging in external optimization engines, various solvers can be used to extend the functionality of the platform. The software functionality is focused on supporting researchers in the algorithm engineering process, facilitating e.g. analysis and comparison of market strategies. An example application of the software to travelling salesman problem, solved by many agents, is presented.

1 Introduction

There exist many drivers for the growth of contemporary computation tasks complexity. The most obvious of them is the increase of data collection, storage and transmission capabilities of todays IT infrastructure. Less evident but equally grave is the fact that isolated systems get more and more connected, and start to exhibit completely different behaviour. Such phenomena are easy to appear, quite easy to expect, hard to foresee, difficult to simulate and mostly impossible to be described analytically. Networking and its intricate couplings has become a fact.

Due to the nonpolynomial complexity of many network problems it is usually very time-consuming, hence impractical, to look for their exact solutions. Let us consider, for example, travelling salesman problem (TSP) whose exact brute-force solving has complexity of $O(n!)$, n being the number of vertices. A much more common approach is to find a good heuristic algorithm, e.g. running in polynomial time [10]. The process of finding a good heuristics is a problem of its own, often consisting of tedious verifications of suites of similar algorithms, varying in details, over a number of test problems, their reference solution known. Managing data from such experiments in order to retrieve useful information can present difficulties, especially if the heuristic algorithm is to be run in parallel.

This paper presents capabilities of a novel software platform for agent-based distributed simulation and its usability for prototyping control strategies. Originally, such control was to be executed over market entities sell/buy offers, in order to achieve optimal allocation of transport network resources. However, it can be equally well applied for coordinating threads of any parallel algorithm, being particularly suitable for graph problems. In such generalized approach market entities become workers processing parallelized tasks, and exchanging data (formerly, 'offers') in a way managed by the coordinator (formerly, 'resource allocator'). The platform imposes a unified problem description and communication scheme, leaving modules implementation details to user's choice — not precluding further parallelism in their implementation.

The next section presents a formal complete model of all objects necessary to describe the process of offering and market clearing, M^3, presented in [2] and exploited by its authors in a number of applications. Then comes the description of the abovementioned software platform facilitating the research on algorithms. Next, the analysis of applicability of the platform in a few more general research scenarios is given, taking as an example the TSP problem solved in parallel by partitioning the set of all possible salesman paths. The paper concludes with studies of more advanced applications of the platform and the underlying problem description format, in research activities.

2 Multi-commodity Market Model

Multi-commodity market model, M^3, is a method and format for a formal description of a market where trade of resources takes place. It has been initially developed to describe offer structure in the energy market in Poland [3]. For its generality it has been next used to model IP network bandwidth trade [4]. With the research platform atop it may be successfully used to model, solve and investigate properties of virtually any graph problem.

M^3 defines the following basic entities and relations between them:

– network nodes and arcs, describing the topology of the network where capacity trade takes place,
– market entities (users, providers) that offer or sell resources (capacity),
– resources being offered, with their proper attributes,
– offers, i.e. bindings of market entities and resources, offered or demanded at a specific price.

It is also possible to define compound resources, i.e. containing simple resources and other compound resources. Analogously, one can define simple and compound offers and market entities. An UML graph representing offers is presented in Fig. 1, to show how flexible the M^3 model is. However, one can use it without being bothered by advanced features, like aggregation facilities. It is possible to declare only key values, *offeredPrice*, *min/maxValue* and *shareFactor* (1 for sell, -1 for buy offers), leaving other unset. The field *acceptedVolume* and *sell/buyPrice* parameters in *Commodity* structure contain results of the market balancing process.

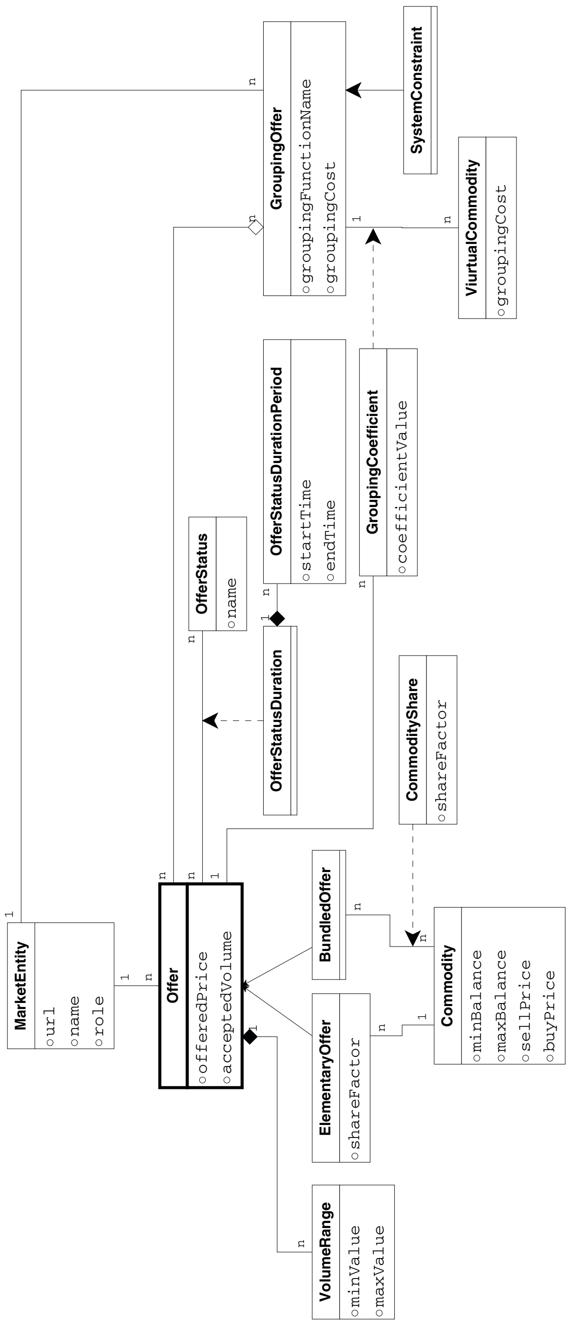


Fig. 1. Relations between offers and other key elements of M^3 model: market entities and commodities. The model allows defining complex offers recursively.

At first, one can perceive the M^3 model merely as a tool for network problem definition, containing placeholders for attributing nodes or arcs with parameters. M^3 defines only the minimum set of those attributes the ones needed in M^3 original applications. Those are defined in XSL dictionary files and, in fact, define the M^3 generic model completely. Concretisation of the generic model for a specific network (e.g. power grid or IP, or road network) is done by specifying extra node/arc attributes in *network kinds* XML file. Analogously, any specific data type describing market entities or offers can be made in proper XML files.

Having concretised the generic type, one can place the data regarding network topology, market entities and offers in appropriate three files. The content of these files defines one market case. Also, it contains placeholders where the solution (resources allocation and resource prices) can be put into. XML has been chosen as a format for M^3 because it makes possible easy transformation of network problems into formats supported by standard optimisation solvers e.g. GAMS, AMPL. This is done by means of XSLT [8].

3 The Platform for Research on Bidding and Resource Allocation Strategies

Good structural design and contemporary technology made M^3 quite popular in the local academic environment. A number of authors used it, mostly in pursuit of developing engines for clearing the market [5], [6]. However, due to variety of approaches to the problem (especially, various assumptions), their results were hardly comparable. It was only recently when they realised that creating a common research environment would be a good idea in several aspects. The major assumption was to store each numerical result in a common database in order to be capable of performing searches, queries and comparisons. This led to work on the formulation of a number of quantitative criteria for ranking the resource allocation algorithms in case of bandwidth trading scenario: economic (total transaction surplus, competition level for network resources) and technical ones (length of allocate network paths, ratio of contracted bandwidth to offered bandwidth) [7].

The platform implementation makes use of many technologies currently being considered industrial standards. The reason for using such standards was to make the platform itself a proof of concept for a to-be commercial trading system. There are three main use cases of the platform:

– Experiment definition. A mechanism designer prepares a set of M^3 files describing the simulation task. Also, the designer implements, if needed, own appropriate resource allocation mechanisms and user agents (or endpoint application for human users acting as agents, cf. Fig. 2).
– Simulation. A chosen experiment is being run: solver and agents' processed are spawned and they communicate alternately with the platform. The results are being stored into the database.
– Result analysis. A selected set of experiment results is selected, by advanced query, from the database. Their results are being compared and displayed in the GUI.

The software was created using technologies allowing application scaling, portability and flexibility. The platform runs as a web service, accessible and configurable via a browser. Such approach minimizes application maintenance costs. To accelerate application operation, some part of GUI processing is shifted to the browser thanks to Google Web Toolkit technology [9]. Links between application modules are managed by Spring framework, allowing for modules replacements without rebuilding of the whole application. The persistence layer is accomplished using recognised products: MySQL, now from Oracle Inc., and Hibernate. The communication with agents and the resource allocation module is managed reliably by Java Message Service. The web application container was Apache Tomcat. Apparently, all the presented technologies are freeware, well supported, popular and documented.

Performance of application three crucial components: XML/POJO serialization, JMS communication and RDBMS communication, has been checked for two test problems. The smaller problem data size was 22 kB, and the bigger problem data was 424 kB. The simulation was run on 4-core PC with Linux OS, using loopback interface instead of a real network. Execution times for different kinds of operations have been logged and analysed, with the following results:

– Serialization times were proportional to the problem sizes (measured in kilobytes).
– JMS communication times were also proportional both to problem sizes and the number of agents.
– RDBMS communication was worse than proportional to the problem sizes, n, but better than $O(n^2)$.

Therefore, the weak points of the architecture were the database and the communication queues. However, implementations of both the technologies come in large variety, and it would be relatively easy to replace e.g. inefficient JMS implementation with another one.

Since the users of the platform will not only run simulations but actually actively develop new agents, a friendly API has been designed, with the aim to keep Java entities structure and communication rules as simple as possible. Any user module, be it agent or solver, must implement methods *onInit* and *onAllocation*, defined by the *Endpoint* interface, requiring and returning *InitParams* and *RunParams*, respectively. *InitParams* contains module configuration data, while *RunParams* carries offers or allocations, depending on communication context.

To relieve platform user from laborious interaction with JMS, a user module skeleton class, *EndpointStub* was created. The class implements main JMS message processing loop, in particular activating *onInit* and *onAllocation* methods in a user-provided object. Also, an exemplary, do-nothing extension of *EndpointStub* was provided to users. The extension, *GenericEndpoint* echoes received parameters of *onAllocation* method; it is excellent for initial configuration testing, and as a base cllass for any user-specific implementation. It also provides a bunch of utility methods, e.g. for XML/POJO conversion.

From the point of view of the agents placing bids for resources, the platform remains rather a transparent thing (Fig. 2). Its only role is to merge individual

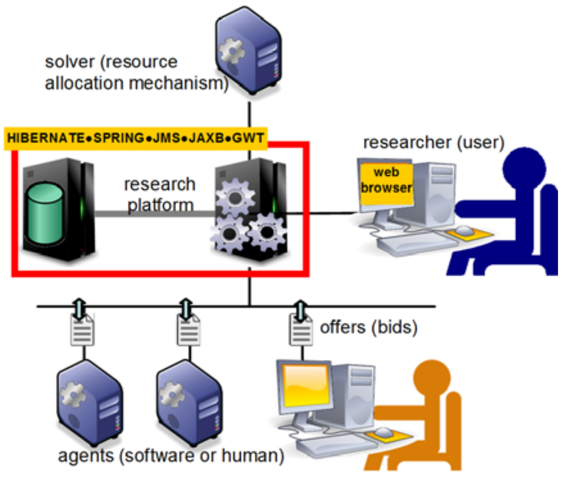

solver (resource allocation mechanism)
HIBERNATE•SPRING•JMS•JAXB•GWT
research platform
researcher (user)
web browser
offers (bids)
agents (software or human)

Fig. 2. Interaction of the platform with bidding agents and resource allocation engine (in simulation mode; message flow is up and down), and in result analysis mode (data flow is from left to right)

requests into a single M^3 document set, which is then presented to the resource allocation module. The content of an exemplary M^3 *offers* file is presented below. Attributes *offeredPrice* contain individual agents' price expectations.

```
<?xml version="1.0" encoding="windows-1250" ?>
<m3:offers xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
xsi:schemaLocation=http://www.openM3.org/m3 M3Offers.xsd
xmlns:m3=http://www.openM3.org/m3 xmlns:ia="http://www.ia.pw.edu.pl/m3">
<m3:Offer id="ia:o12345-67" offeredPrice="32000.00">
  <m3:description>Exemplary elementary offer</m3:description>
  <m3:offeredBy ref="ia:siekierki" />
  <m3:offerStatus status="m3:offer-open">
  <m3:durationPeriod startTime="2007-04-09T08:00:00"
    endTime="2007-04-09T09:59:59" />
```

Resource allocations are then sent back to the bidders (using *Commodity* section of M^3 model) and, optionally, the process is re-iterated until the allocations settle. This could be done easily without the platform altogether, save for the fact that all bids and allocations get stored in the database both in plain XML and in the structured form. Turning the original XSD definitions into an object hierarchy and a RDBMS scheme constituted the biggest part of the architectural and programming work. The information flow for the most important use case, the simulation, is presented in Fig. 3.

Once stored in the database, experimental results can be compared easily in a number of ways. A flexible system of experiment tagging allows searches across many dimensions: the author, allocation engine type, version and running

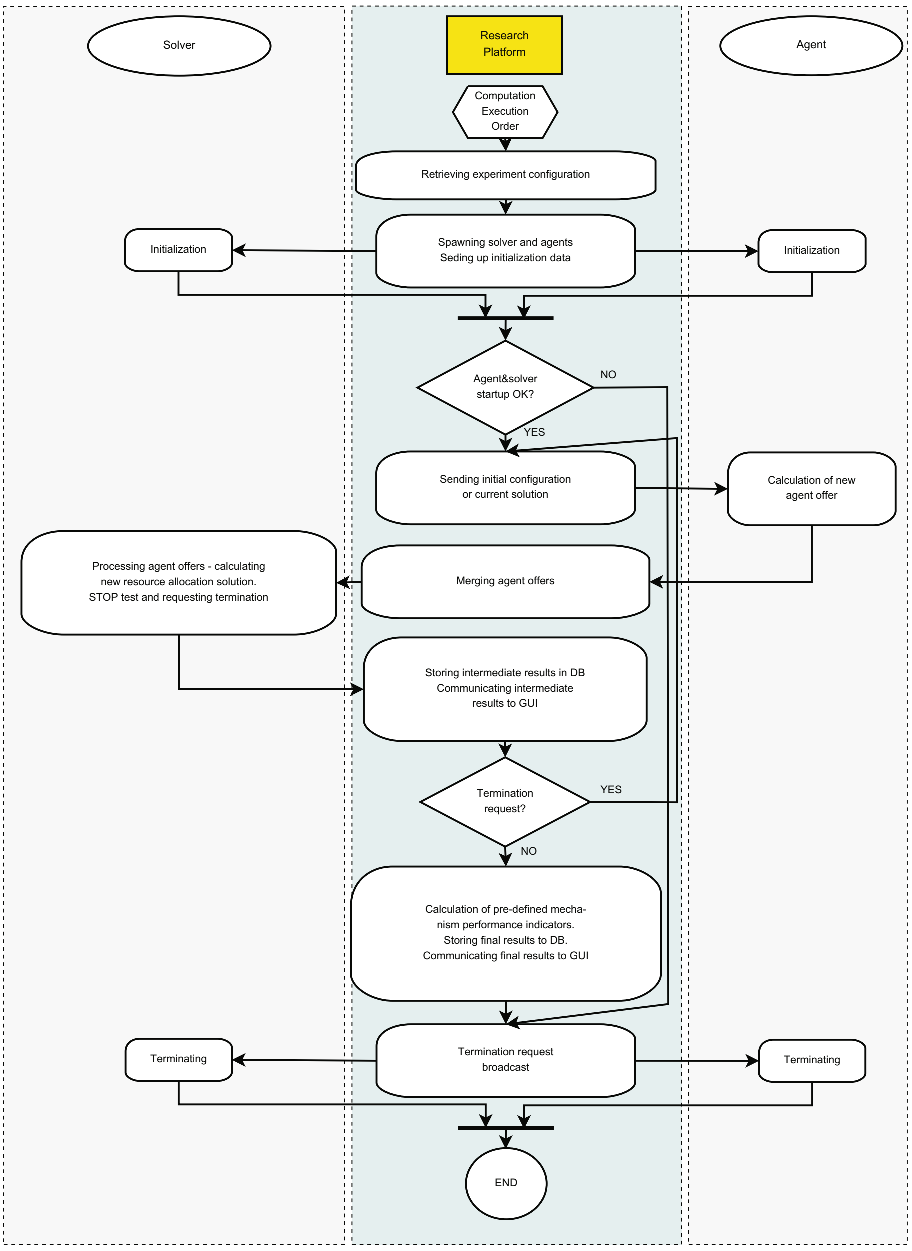
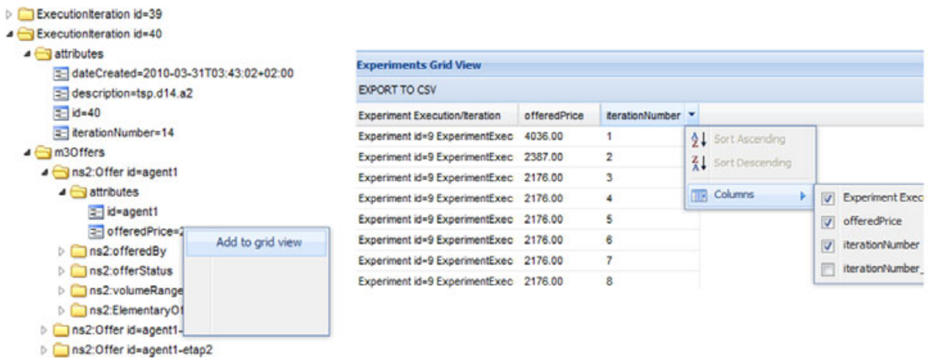

Solver
Research Platform
Agent
Computation Execution Order
Retrieving experiment configuration
Initialization
Spawning solver and agents Seding up initialization data
Initialization
Agent&solver startup OK?
NO
YES
Sending initial configuration or current solution
Calculation of new agent offer
Processing agent offers - calculating new resource allocation solution. STOP test and requesting termination
Merging agent offers
Storing intermediate results in DB Communicating intermediate results to GUI
Termination request?
YES
NO
Calculation of pre-defined mecha-nism performance indicators. Storing final results to DB. Communicating final results to GUI
Terminating
Termination request broadcast
Terminating
END

Fig. 3. Data flow diagram for the simulation use case



```
ExecutionIteration id=39
ExecutionIteration id=40
  attributes
    dateCreated=2010-03-31T03:43:02+02:00
    description=tsp.d14.a2
    id=40
    iterationNumber=14
  m3Offers
    ns2:Offer id=agent1
      attributes
        id=agent1
        offeredPrice=2
      ns2:offeredBy
      ns2:offerStatus
      ns2:volumeRange
      ns2:ElementaryOf
    ns2:Offer id=agent1-
    ns2:Offer id=agent1-etap2
```

Experiments Grid View

EXPORT TO CSV

Experiment Execution/Iteration	offeredPrice	iterationNumber
Experiment id=9 ExperimentExec	4036.00	1
Experiment id=9 ExperimentExec	2387.00	2
Experiment id=9 ExperimentExec	2176.00	3
Experiment id=9 ExperimentExec	2176.00	4
Experiment id=9 ExperimentExec	2176.00	5
Experiment id=9 ExperimentExec	2176.00	6
Experiment id=9 ExperimentExec	2176.00	7
Experiment id=9 ExperimentExec	2176.00	8

Fig. 4. Screenshots of results browse tree (left), with possibility to indicate variables to be exported or displayed as a table (right)

parameters, type of the problem etc. A number of specific data can be then extracted from such pre-filtered data: allocation or prices of a resource in n-th or last iteration, total number of iterations etc. Those can then be arranged freely in a table in order to draw a graph or to be exported for further processing outside the platform (Fig. 4). A number of pre-defined market mechanism performance indicators are calculated in each simulation iteration, and stored in the database. They include the total benefit of the operator of the market as well as a similar coefficient defined for the market participants. New algorithms for scoring simulation results, taking into account specifics of individual problems, can be easily implemented. Such freedom is also well handled by the database design, allowing any number of so obtained measures. Apart from the compiled-in functionality, the platform provides means to formulate and evaluate *XPath* expressions over some queried result set of performed experiments. In such way, platform user may calculate fancy measures for experiment results.

4 Application of the Platform to Parallel Computation Tasks

Inherently, the presented platform is able to support both distributed computation and subsequent analysis of results for any problems describable in the M^3 model. This means it can handle most of graph problems: TSP, coloring, transport etc. Also, it can be applied to massive parallel algorithms that are apparently not related to graphs, as evolutionary strategies. Noteworthy is the possibility of performing social networks analysis, using the platform. The most computationally demanding part of extracting knowledge from the social graph is to find communities, i.e. sets of vertices particularly mutually closely connected, in a sense. The complexity of exact solution finding is *NP*-hard. That is why approximate community-finding algorithms are being constructed. The platform can easily improve the process of good heuristic algorithm research in the field. This application scenario has been presented mostly for current popularity of social network analysis, applicable e.g. in marketing process, fraud detection, business/military/terrorist organisation structure analysis. Social network analysis is another topic of interest of the author of this paper, and the prospect of employing the platform in such research activities will hopefully be fruitful.

The platform has been successfully applied for an exemplary travelling salesman problem. The purpose was to demonstrate that coding of the problem can be simple, and that using the environment does not require any clumsy workarounds. One starts with adapting basic M^3 definitions in order to create data types matching ones problem. In our case no changes to the built-in market entity and network definitions were needed. The M^3 "commodity" term has been used in TSP problem to represent a single stage of a salesmans route. An extra property *stage_number* was created to identify stages. All stages are linked to an artificial network node, *n0*. Recall that offers tie stages and market entities. In the context of TSP, market entities are identical with computation units performing the work in parallel. To code the actual solution found by a unit, standard *minValue* and *offeredPrice* fields of the offer type. The first one indicates the concerned stage, while the second one indicates the node a salesman passed in that stage.

Computation units are started in this example as Java instances: they perform a specified part of the computational task, and report progress to the platform periodically. The resource allocation module implementation is a trivial one, selecting and recording the best solution found so far by the agents.

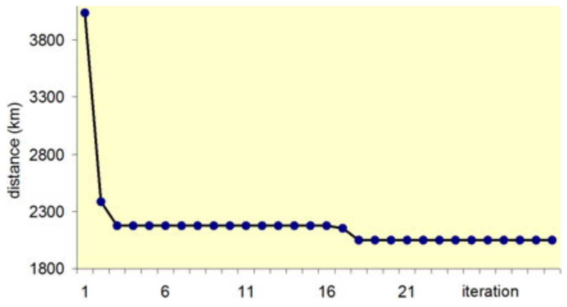


Fig. 5. TSP solution in subsequent algorithm steps

The example problem was defined for 14 major Polish cities. The graph of connections was full mesh, with geodesic distances. Rather than coding those distances as M^3 graph arcs, there were calculated dynamically by each agent using a dedicated Java package [1]. Thus, the M^3 network definition consisted only of cities names, latitudes and longitudes. No arcs were declared in order to accelerate communication. The exact solution of the problem was found correctly

(Fig. 5). There were two path calculation modules performing, one resource allocation module and the platform software, of course. In the case described no relevant information was passed back from the allocation to computation modules; however, passing some extra control data, e.g. adjusting their operations, would be easy to implement.

5 Conclusion

It was shown that the platform originally developed for investigating properties of resource allocation problem to the market can be successfully used to support research also in other areas. The presented example was a TSP problem. Platform's ability to incorporate external solvers and the design based on M^3 information model make it a convenient tool to perform comparative analysis of parallel tasks — a tool for a groupwork research.

The presented platform API was designed in abstraction of any programming paradigm. In fact, it does not impose any constraint on the user, save for the necessity of implementing the two methods: *onInit()* and *onAllocation()*. In particular, the user has the ability of accessing any object of the M^3 structures directly, by using POJO representation of the XML code. Alternatively, she or he may prefer to do high-level XSL transforms on the plain XML to complete problem definition with extra information (e.g. the objective function), difficult or impractical to be stored in M^3. Then, an external solver can be called. Also, the possibilities of performing any local parallelisation of the pre-assigned piece of the problem, are not limited in any way.

It should be stressed that the added value of the work presented in this paper is not related to equipment or algorithmic performance. Most of the work presented here was spent of mapping advanced modelling scheme, M^3, into a relational database, and on developing ergonomic programming and graphical interfaces, with the aim to create a teamwork scientific environment. Hopefully, this will make cooperation quicker and clearer, and the results — more objective.

Acknowledgments. The work presented in this paper was supported by the Polish Ministry of Science and Higher Education research grants no. PBZ-MNiSW-02/II/2007 and N N514 416934.

References

[1] Gavaghan, M.: GPS Receivers and Geocaching: Vincenty's Formula — a section of the web page, `http://www.gavaghan.org/blog/category/codeproject` (accessed March 30, 2010)

[2] Kacprzak, P., Kaleta, M., Pałka, P., Smolira, K., Toczyłowski, E., Traczyk, T.: M^3: Open Multi-commodity Market Data Model for Network Systems. In: Proceedings of the XVI International Conference on System Science, Wrocław (2007)

[3] Kacprzak, P., Kaleta, M., Pałka, P., Smolira, K., Toczyłowski, E., Traczyk, T.: Communication model for M^3— Open Multi-commodity Market Data Model. In: Proceedings of TPD 2007 Polish Conference, Poznań (2007)

[4] Kacprzak, P., Kaleta, M., Pałka, P., Smolira, K., Toczyłowski, E., Traczyk, T.: Application of open multi-commodity market data model on the communication bandwidth market. J. Telecommunications and Information Technology 4, 45–50 (2007)

[5] Karpowicz, M., Malinowski, K.: Network flow optimization with rational agents. NASK internal report (2009)

[6] Pałka, P., Kołtyś, K., Toczyłowski, E., Żółtowska, I.: Model for Balancing Aggregated Communication Bandwidth Resources. J. Telecommunications and Information Technology 3, 43–49 (2009)

[7] Stańczuk, W., Pałka, P., Lubacz, J., Toczyłowski, E.: Parametric pricing rule in bandwidth trade. In: Proceedings of 8th International Conference on Decision Support for Telecommunications and Information Society DIST 2009, Coimbra (2009)

[8] XSLT transformation "toAmpl-BCBTxsl" Documentation contained in archive available in Tools/XSLT files section of the web page, `http://www.openm3.org` (accessed March 30, 2010)

[9] Google Web Toolkit Overview, `http://code.google.com/webtoolkit/overview.html` (accessed March 30, 2010)

[10] Ausiello, G., Leonardi, S., Marchetti-Spaccamela, A.: On Salesmen, Repairmen, Spiders, and Other Traveling Agents. In: Bongiovanni, G., Petreschi, R., Gambosi, G. (eds.) CIAC 2000. LNCS, vol. 1767, pp. 1–16. Springer, Heidelberg (2000)