Abstract—This paper describes an implementation of a communication framework for power systems simulation. The framework is based on the principles of the multi-agent systems.

The framework is based on technologies which became de-facto standards in this scientific field – the communication protocols CORBA (Common Object Request Broker Architecture) and KQML (Knowledge Query and Manipulation Language). The system itself economizes the open source implementation of the protocols.

The whole system is dedicated to active simulation of the energy flow in a power system and its visualization. The system design takes into account possible future replacement of any particular agent with an on-line power equipment monitoring facility (ad-hoc sensors), which allows to monitor the whole power system in real time.

I. INTRODUCTION

The fault-tolerance and quality assurance of the power supply is very important and constantly evolving science branch which demands intensive research. Because of the geographic dissipation and big financial cost of the power systems facilities maintenance, the required reliability cannot be reached simply by redundancy. It is necessary to search for the new low-cost but effective means of the required reliability assurance.

One of the possible approaches to this problem is regular control of the facility operation. Failure data are gathered in failure databases with a precise description of the failure cause, time, severity and the description of the failed component – its manufacturer, model, serial number etc. This effort allows a statistical analysis of the reliability of the particular facility series, types etc.

This type of the fault database is for the whole Czech republic built and maintained by the research team for creation and categorization of failures records for distribution equipment and outages of supply at all voltage levels in the Technical University of Ostrava. Utilities for the analysis of the data in this database are also developed there. The aim of this effort is to develop a new methodology for the power system facility maintenance based on the condition-centered approach rather than the contemporary used preventive time-based approach. Power system facilities are well tested in the process of their design and manufacturing but there’s a serious lack of testing in the process of their real operation. This project tries to overcome this lack.

The new methodology should be enough flexible to allow a fast and effective control even for facilities that lack measured reliability data. The Technical University of Ostrava with the help of other universities in the Czech Republic has made an interdisciplinary team which is focused on the development of new methodology for the power system facility condition checks, outage prediction, black-out risk reduction and early post-fault recovering.

The team at FI MU Brno is focused on the usage of the multi-agent systems in the power system facility simulation, monitoring and control.

II. REQUIREMENTS

The design of our multi-agent framework is based on the following basic requirements:
1) decentralization
2) platform independence
3) performance scaling
4) modularity and extensibility
5) open standards
6) security
7) low-cost practical application

A. Decentralization

Because of the power systems nature (especially their geographic dissipation), the system allows decentralized distributed operation on many computers interconnected with the standard type of computer network (LAN, WAN, Internet). This requirement is even more important for the process of real-time on-site monitoring of the facilities.

B. Platform independence

The application environment is very heterogeneous. It is absolutely necessary to use technologies independent on hardware platform and operating system. The framework should also support wide range of programming languages.

C. Performance scaling

Hardware requirements of the simulation and a range of the future applications is hardly predictable. The framework therefore supports easy performance scaling by extension of hardware and software dedicated to it. This extension should not require large changes to the existent architecture and implementation.

D. Modularity and extensibility

The framework is flexible enough, open and extensible for the addition of new features and modules whose need appears in the process of dissemination and testing. Even in this case, changes to an existing code should be as minimal as possible.

E. Open standards

We must take into account a possible future need to integrate the proposed system with the contemporary software and hardware solutions of the power system facility vendors. To make this integration easier, the system is based on standard technologies – with open specifications or even implementations.

F. Security

Communication between system modules usually (at least in some cases) goes through untrusted channels. Privacy and authenticity of the particular communication acts must be still guaranteed on the level common in the industry solutions.

G. Low-cost practical application

The framework is based on the commonly used technologies. Only then we can assume an easy and low-cost practical application in the production environment. If any usage of non-common technologies will prove to be necessary, it should not appear in many parts of the system, especially not in the user-oriented parts of the system.

III. System architecture

After the specification of all the system requirements (see [1]), we have decided that the best results will be acquired by implementing the system as a multi-agent system based on the standard technologies commonly used in this area – a combination of two communication protocols, CORBA and KQML. CORBA standard has freely available specification and there are many implementations – proprietary ones (e.g. Visibroker, PeerLogic, IONA) and even free ones (e.g. OmniOrb, Orbit, Mico). In our prototype implementation, we use OmniOrb (in Python) and Sun ORB which is part of the Java SDK. An indispensable advantage of CORBA standard is that all ORB implementations conforming to the specification should integrate well within the system.

Usage of the CORBA-KQML solution was many times published in the literature (e.g. [4], [5], [6]). The multi-agent approach has been used even in the area of the power systems monitoring and control (e.g. [7], [8], [9]).

As discussed in [8], old “hardwired” solutions of the power systems monitoring have many problems caused by their limited flexibility which we are going to overcome by using the open and extensible multi-agent system implementation. This approach will be strong enough to satisfy all our requirements.

Our proposed communication architecture is made up of many autonomous and self-deciding agents (see the Figure 1).
A. General agent characteristics

A general description of an agent-based software engineering methods, that form the basis of the presented system, can be found e.g. in [2]. Case studies of actual applications with agent-based architecture in the industrial control systems are presented in [3].

The main principle of our system is that each agent can be implemented by a stand-alone process or even stand-alone computer in a computer network. If there is a huge amount of communication between agents, it is possible to implement two or more agents within one process and thus lower the communication overhead. This change needs only small intervention into the code (see Section V).

All agents in the system have the ability to communicate through CORBA and KQML (see the Section IV). Each agent has a defined type (denoting a set of messages which are accepted and understood by the agent) and it is identified within the network by its distinct name and a given identification number. The agent types are implemented as a hierarchy in which each level is assigned a set of mandatory KQML messages which every agent of this type must understand and must be able to respond to it (this is similar to the inheritance in object programming languages). The hierarchy looks like that:

- Agent
  - Real-life agent (represents a particular power system facility)
    - Line
    - Facility
      - Source
      - Transformation station
    - Switching station
    - Consumer
  - Organizational and auxiliary agents
    * Register
    * Helper
    * Viewer

Each agent that belongs to some category must be able to respond to every message from the defined set of messages. For example, agent of the type Source must be able to respond to all messages mandatory for types Source, Facility, Real-life agent and Agent.

1) Register: The Register agent holds a database of all agents in the system, their name, type and Agent ID (AID). Every agent must notify the register about its existence before joining the network. The register then passes a communication key to the agent. This approval for the network joining serves as the certificate of authenticity and can be validated by each agent in the network. The key also plays a role of an encryption key for securing inter-agent communication. Agent is also given an AID. The register agent can return an agent network address as respond to query based on agent AID, name or type.

2) Helper: The Helper agent implements all global functions that are not bound to any particular agent but are relevant to the system as a whole – e.g. system identification values, supported protocol versions, description of the allowed agent types, global configuration constants etc.

3) Viewer: The Viewer agent performs a graphical display of the events that take place in the system. There can be plenty of viewer agents and each of them can visualize the state of the system in a different manner – e.g. to plot a graph or to save the data in a database. Viewer agents joining of the network is performed strictly dynamically – after joining, the Viewer asks the Register for notifying about network addresses of all viewable agents (KQML performative subscribe) and then asks each of the agents to constantly provide information needed for visualization of the agent (the same performative).

4) Real-life agents: In the sense of the multi-agent systems theory, each part of the system is an agent. By the term “real-life agent” (RLA), we mean an agent representing some particular power system facility (it has an equivalent in the real life).
RLAs has (as opposed to other agents) a huge set of mandatory messages regarding the power flow description.

5) Line: A power line is the only part of the network which builds up the network topology (i.e. its interconnection). Each line is labeled with names of the two agents representing its input and output. Other RLAs (of the facility type) are passive – they wait for the connection and do not initialize it in any manner. So the topology of the RLAs remains stable in the whole run which corresponds to the reality of the power systems. On the other hand, the connection with other agents is dynamic – it is possible to assign new monitoring agents to a running system.

IV. AGENTS COMMUNICATION

The basic assumption is that agents can be located on separate machines or at least separate processes within one machine. Hence the communication between them must be constituted by some kind of network protocol – in our case it is TCP/IP. But this protocol represents only the lowest layers of the communication which is extended by three other protocols on the top two levels of the OSI model [10] – they are CORBA, KQML and the content language itself.

A. CORBA

The inter-agent message-transporting layer is constituted by CORBA (Common Object Request Broker Architecture [11]). This technology implements a transparent middleware for data transport and function calls between processes in one machine or two separate machines. All operations all strictly platform independent – in the sense of hardware and software equipment of the machines. Moreover, there are very good standard specifications of a data-structure mappings into many programming languages (we are using mappings into Java [12] and Python [13]).

A mapping into particular language is possible because all data structures and object interfaces are defined with a neutral meta-language IDL (Interface Definition Language [14]). This language is formally similar to structures used in data modelling meta-language UML (Unified Modelling Language).

For instance, a definition of a data structure and an interface of an agent which is able to communicate in KQML looks in IDL this way:

```idl
struct KqmlMessageStruct {
    string performative;
    long sender;
    long receiver;
    /* ... */
    any contents;
    long inReplyTo;
    string force;
};

interface KqmlReceiverInterface {
    exception badKQMLMessage{};

    // send KQML message - standalone args
    void kqmlMessageArgs(in string performat,
        in long sender, in long receiver,
        /* ... */
        in any contents, in long inReplyTo,
        in long force)
        raises (badKQMLMessage);

    // send KQML message - struct
    void kqmlMessage(in KqmlMessageStruct message)
        raises (badKQMLMessage);
};
```

For each of the supported languages, there is an IDL compiler which builds a skeleton in the desired language. The skeleton corresponds to the structures defined in IDL (in the above example, it will build in Java language a KqmlReceiverInterface interface). The big advantage of CORBA is the fact that objects communicate with each other in the same way as when they are in the same process – CORBA compliant ORB (Object Request Broker) facilitates all data transformations and network transportation absolutely transparently. This feature is especially important for possible system rearrangements for the performance enhancement (see further).
Another very important feature of CORBA which is used in our framework is so-called CORBA Naming Service. This service is similar to the Internet Domain Name Service, DNS. CORBA Naming Service converts an object name to a distinguished network address. Thus the knowledge of an agent name and an address of the CORBA NS server is sufficient for establishing a connection with the agent. NS server will give an agent address on demand. This way it is possible to form an architecture which can be subject to changes and is absolutely independent on the part of network the agent is physically running. This key feature makes it possible to reach the required system flexibility. Agents need to know only the names of agents they are about to communicate with – all other information is gathered dynamically.

B. KQML

Under the CORBA layer which facilitates the connection between agents, there is a KQML (Knowledge Query and Manipulation Language [15]) layer. The KQML language is based on the linguistic theory of the speech act [16], published by Searle in [17]. This theory in short says that every communication act can be categorized as for instance: an announcement, a query, a demand etc.

KQML communication is strictly divided into two levels – level of the speech act resolution and the message content resolution. For each type of the speech act there is one or more so-called “KQML performatives.” Performatives give us a basic information about what type of information or action an agent demands.

KQML language is defined as an abstract query and manipulation language. The basic concept is that each of the communicating agents has its own knowledge database concentrating knowledge about the outer environment and the agent itself. But this is only a formal concept. In reality the agent can work even strictly reflexively – but it must represent its actions as a manipulation with his or others knowledge database. For this reason we call the desired data structure on which the KQML operates a Virtual Knowledge Base (VKB).

KQML has two fundamental concepts: (1) each agent is autonomous and only it decides what to do in a particular time, (2) agent A can ask agent B for information from its VKB. According to these information and according to the content of its own VKB the agent drives its behaviour. KQML contains a set of performatives which express a desire of an agent A for agent B to make an effort to achieve some state of the environment. These performatives can be used for agent activity control but at present time they are not used in our framework.

KQML message does not define a form of the message meaning itself. It only defines from whom, to whom and when was the message delivered and in which form the sender expects an answer. The content itself is communicated by so-called content language. The content language can be any possible language which is adequate for the message content expressing and which is understood by both the agents. In practice, usually PROLOG and KIF (which was designed especially for this purpose) are used.

C. Example

For an illustration of the inter-agent communication we introduce a typical situation in a power system simulator (the example is simplified for better clearness): agent A represents a transformation station transforming a very high voltage to a high voltage. Agent A is connected to agent B which represents high voltage line and which is connected to agent C representing a distribution point (see the Figure 3).

Agent B asks agent A for a notification for every A’s state change and every A’s output voltage change:

```
{subscribe
 :sender B
 :receiver A
 :timestamp 1113340454
 :reply-with query_1
 :language KQML
 :ontology KQML_ontology
 :content
  (ask
   :sender B
   :receiver A
   :in-reply-to query_1
   :reply-with 2
```
As soon as the agent forms a KQML message, it contacts a CORBA Naming Service and asks it for a network address of the recipient (in the case that it is not in his VKB yet) and contacts the recipient (again, in the case that it has not done it already before). The sending of the message itself is realized by a simple CORBA call which is in the formal sense identical to a call which would represent a call of a function in the same process (programming language “native” call).

By this technique a dynamic connection between the agents A and B is established (i.e. the output of device represented by agent A is connected to the input of the device represented by the agent B). Similar connection is established between agents B and C.

If there is an outage on line B and the current supply is canceled, B sends to C a message of this format:

```
{tell
  :sender B
  :receiver C
  :timestamp 1113341454
  :in-reply-to query_2
  :reply-with query_3
  :language Prolog
  :ontology Power_system
  :content out_voltage(0),
       state(fatal_failure)
}
```

C reacts to this message in this way: it sets all its outputs voltage to 0 V too or switches to another (backup) input line.

The big advantage of KQML is that the agent B can send a message with the same format to the visualization agent or to an agent which loads data into the failure database. It is not necessary to develop a specialized protocol for each communication channel even if the content of a message is somehow different.

V. PERFORMANCE QUESTIONS

A. Data preprocessing

A quality agent design should be based on the following principle: “Agents are communicating only those data which they really need” – i.e. maximum amount of data is processed locally by the agent. The other agents should ask only for results of this processing. E.g. the running temperature of a facility can significantly affect the reliability of the facility but no agent should ask for the temperature – instead it should ask for the condition of the facility or the probability of a failure in following time period. The amount of the communicated data can be significantly reduced in this way.

B. Choosing the adequate content language

Content language should be a language which is a “native” language of an agent. It is very inefficient to choose a language which demands a conversion of an existing data to some other format and back just for the reason of communication.

C. Direct calls

In the cost of lower flexibility of the KQML format, it is possible to send some time-critical data by a direct CORBA dedicated function calls or even join two or more agents into one process and use the function call native to the implementation programming language. It is possible to gain a significant performance improvement in this way, but it means to “hardwire” such an implementation. This requires a deep intervention into the existing code and thus should be considered a real makeshift.
On the other hand the CORBA transparency requires only minimal change of the code in the case that existing CORBA-enabled functions (i.e. functions defined in the IDL) are changed to direct native calls without ORB. The possible scenario of a communication without the KQML layer is illustrated in the Figure 4.

D. The change of a physical location

The network transportation layer is the main reason of the communication delays. This overhead can be overcome if the communicating agents are located on the same machine or at least on the same network segment.

VI. CONCLUSION

The described communication framework follows all the required criteria which were identified in the process of power systems simulation analysis. It has already proven to be enough flexible, open, dynamic and robust even for possible real-time power systems monitoring. This is continuously validated by the growing implementation and it will be tested in cooperation with large Czech power supply companies.

The system is general and adaptable enough to be applied not only in the case of a power system simulation. Namely, its architecture is suitable for usage in a largely decentralized networks of autonomous data gathering and processing units. Especially an integration with networks of thin-client sensors (e.g. ZigBee [18]) deserves further attention because of the promising industrial applications.

ACKNOWLEDGEMENTS

This work has been partly supported by Grant Agency of the Academy of Sciences of CR under the project T100300414 “Intelligent methods for increasing of reliability of electrical networks.”

REFERENCES