Computing and Informatics, Vol. 28, 2009, 811–842

# ANALYSIS OF THE ERROR PROPAGATION PHENOMENON IN NETWORK STRUCTURES

Dariusz Król, Grzegorz Kukla

*Institute of Informatics*
*Wrocław University of Technology*
*Wyb. Wyspiańskiego 27*
*50-370 Wrocław, Poland*
*e-mail:* `Dariusz.Krol@pwr.wroc.pl`

Revised manuscript received 19 January 2009

**Abstract.** The analysis of error propagation is of fundamental importance to assure safe operation and management of abnormal situations in any distributed information system. In this paper, the quantitative and qualitative methods are proposed to analyze possible error propagation scenarios based on different topologies, error types and probability distributions. The most interesting from our point of view is the course of error propagation in simple structures that are contained in more complex ones. These complex structures, which have attracted the attention of scientists for many decades, are traditionally analyzed with the use of formalisms from graph theory. Certain types of graphs are often used to model naturally occurring complex structures, such as social networks. Graph-theoretic approach proved successful when applied to social networks and other naturally occurring complex networks. The research was verified based on the experiments conducted on simulation model. The results provide some ideas of robustness – the knowledge how to design the most error resistant architectures in complex environments.

**Keywords:** Distributed information systems, dependable computing, web dynamics, semantic network, social network, network topology

## 1 INTRODUCTION

The study on error propagation belongs to the common area of different software engineering disciplines. Preventing spreading of faulty behaviour is in the scope of interest of data propagation, dependability assurance and safe algorithm design. Despite its importance, the problem of error propagation prevention, e.g. in social structures has not been a subject of a separate thorough research. The design of distributed systems takes the presence of faults into account, but nonetheless the nature of error propagation phenomenon is not well examined and properly described. For that reason the study on error propagation can help in implementing complex dynamical modern structures correctly and in placing them in the proper place of the system.

There are several research directions for error propagation phenomenon. The first one is the concept of an error's scope, defined as the portion of a system that it invalidates [22]. The second one is enabling a developer to make better use of the exception propagation paths, which includes the origin of exceptions, handler and visualization tool [4]. The investigation of the performance metrics to enable quick evaluation of distributed systems, also error propagation, without reorganization of the running system can be found in [9]. Nikoletseas and colleagues [15] considered a model for intrusion and its propagation through various attack schemas. Lin and Kuo [10] describe a state preservation scheme to find the optimal recovery points of the processes suffering from error propagation. Another approach to propagation presented by Martel in his work [14], where the propagation of roundoff errors throughout a calculation is introduced. In our previous work [8], we addressed the problem of an automatic framework for efficient propagation of class package and data. Fault propagation analysis (FPA) is the most popular to assure safe operation and for the management of abnormal situations in plants. In the paper [6], practical framework to synthesize and assess all possible fault propagation scenarios based on robust modeling is proposed. To understand the mechanism for FPA, a detailed semantic network with process variables, equipment variables and rules is designed.

Our current work differs from previous ones in that the previous works focused on the benefits of data propagation rather than on analysis of abnormal situations like faults. Now, we can estimate error propagation probability in complex topology while others estimate scope and recovery methods rather than the quantitative and qualitative analysis of faults. Moreover, our method can predict the weakest points in the system.

The problem of error propagation is present in many domains of computer science. However, according to the knowledge of the authors, not much focus has been spent on the phenomenon itself. Most papers related to reliability or fault tolerance just state that errors can spread across the system and this process needs to be reversed in order to restore its operation. Nevertheless, not much focus is put on explaining how exactly error propagation proceeds. As stated earlier, studies on error propagation can provide useful results for designers, developers and software engineering researchers. This work tries to fulfill the gap in research on error propagation phenomenon in network structures.

The rest of this work is organized as follows. Section 2 presents the state of the art in the domain of real-world structures. Section 3 concerns common topologies and error types between entities. Section 4 describes the experimental environment used for hypothesis verification. It provides the information about the structure of the utilized model and principles of experiments. Section 5 presents the results gathered during experiments and their analysis. The last section summarizes the work and provides conclusions.

## 2 NETWORK STRUCTURES

Real-world networks are generally very far from random structures [16]. Recent research has revealed that e.g. social structures have a much richer structure than regular or random networks, hence we have small-world and scale-free topologies. The types of networks analyzed in these studies include the Internet and WWW, scientific collaboration networks, gene interaction networks, various social networks, and neural networks. Although much research in artificial intelligence has focused on robustness in distributed systems, little effort has been devoted to understanding that modern structures behave not the same way when error occurs.

Recent findings highlight the importance of social structure on organizational performance for various types of artificial societies. Such studies have shown that clustered organizations, where objects have few connections but are tightly connected, are more stable than organizations with high connectivity [7]. In the context of virtual team formation, networks that exhibit short average path length allow for greater diversity in teams as well as efficiency in forming teams. Sakata [18] reported topological features in social networks and topological similarities between mammalian cortical networks and social friendship networks. Milo et al. [13] found motifs, patterns of interconnections occurring in complex networks at numbers that are significantly higher than those in randomized networks, from biochemistry, neurobiology, ecology, and engineering.

Real networks, which have attracted the attention of scientists for many decades, are traditionally analyzed with the tools of graph theory using more often the average node degree, the average path length, and the clustering coefficient. Networks constructed from real-world data share common global properties [2]. First, they are typically sparse. The vast majority of nodes are connected only to a small percentage of other nodes, and the number of edges is closer to the number of nodes than to the square of the number of nodes. Second, they have a short average path length and strong local clustering. The neighbors of a given node are more likely to be connected to one another than to be alone. Random graphs, by contrast, have a short average path length but a low clustering coefficient. Third, the distribution in node degree is characterized by a power law. This means that there are a small number of nodes which have many neighbors and a large number of nodes that have only a few neighbors. The second and third properties are of special note, because they are the signatures of small-world and scale-free characteristics, respectively.

In this section, several network models are briefly surveyed with an emphasis on models that attempt to reproduce the properties of networks found in the real world.

**Regular Structures.** Regular graphs are best characterized as having a homogeneous connectivity pattern for all of the nodes in the graph. In these graphs, the degree distribution is trivial: all nodes have exactly the same degree. Examples of regular graphs include lattices, hyper-cubes and fully connected networks (i.e., all nodes are connected to all other nodes). In lattice graph topologies, nodes are given a logical ordering and connections among nodes are limited to nodes that are within close logical proximity. The dimension of a lattice determines the number of coordinates required to specify the node positions. In order to prevent boundary conditions (i.e., different node degrees for the nodes on the boundaries of a given lattice), a lattice can be wrapped upon itself. In one dimension, wrapping a lattice results in a ring topology.

**Random Structures.** Recent evidence suggests that random graphs are not representative of real networks. They do possess short average path lengths. On the other hand, random graphs do not exhibit clustering or a power-law degree distribution (the degree distribution follows a Gaussian distribution).

**Small-World Structures.** These are highly clustered graphs, like regular lattices, with small characteristic path lengths, like random graphs. This is called the small-world property. A key observation is that small-world networks have properties that lie between those of regular networks and random graphs. Lattices are the basic building block of small-world networks. In graphs with small world properties, there are highly clustered neighborhoods and it is possible to move from one node to another in a relatively small number of steps (often just two or three, on average.) As a small-world graph grows, the average path length increases slowly, as a function of the logarithm of the size of the graph. This is in opposition to the longer path lengths of a regular grid-like network in which hundreds or thousands of steps may be required. Watts [23] and Strogatz [21] presented that two properties, short paths and high clustering, would hold also for many natural and technological networks. The intuition is that the short paths could provide high-speed communication channels between distant parts of the system, thereby facilitating any dynamical process, like synchronization or computation, that requires global coordination and information flow.

**Scale-Free Structures.** Scale-free networks [1], which include the web-like networks [3], peer-to-peer networks [11], some social networks and cells have not the small-world property. The scale-free graph model is motivated by the empirically measured degree distributions of the Internet and the WWW. The model is a highly intuitive model based on how networks are believed to evolve and grow in the real world. The generation of scale-free graphs has two simple rules: a) growth – at each time step, a new node is added to the graph, and b) preferential attachment – when a new node is added to the graph, it attaches preferentially to existing nodes with high degree. Scale-free networks have no characteristic scale of node degree; instead, they exhibit all scales of connectivity simultaneously. As a result, they tend to have a similar appearance when examined at varying scales. They are also generally robust against random disruptions; if any node is removed at random, the statistical likelihood that the node will be a hub is low. It is far more likely instead that the node selected for removal will have a small number of neighbors. Therefore, a random change is not likely to have a significant effect on the structural form. By contrast, if hubs are selected for removal, the topology of the network will undergo a significant change, since the hubs are connected to a large number of neighbors.

This set of models was chosen because they are widely studied, highly intuitive, and model either the structure of real networks, or have been traditionally used to model the interconnectivity of network systems.

According to the literature discussing network structures the connectivity of the system's topology defines the boundaries of the number of possible errors that can be tolerated by the system. This indicates that the system's physical construction affects its behaviour under its element's failure circumstances. However, the usage of the connectivity measure is not sufficient to define the exact influence of topology on error propagation's course. This measure oversimplifies the reasoning, because it reduces completely different constructions into one class. The available literature, like [19], claims that the higher connectivity of the system's communication graph, the higher error tolerance of the system. On the other hand, one cannot disregard the connections structure of the system, because the scope of the error is strictly bounded to it. The hypothesis that needs to be verified by this work states as follows: the topology of the network system has the influence on the error propagation problem occurring within it and different topologies having the same connectivity can behave differently.

## 3 STRUCTURES APPLIED IN RESEARCH

### 3.1 Topologies in Information Systems

The way the elements of a distributed information system are organized is called a topology. The topology of a system determines the communication patterns utilized within it. It must be emphasized that topology has the influence on the logical connections between system's elements. Not every physical link between the elements of the system is utilized during messages interchange. Therefore, the topology must not be confused with the physical organization of the system, such as its network structure. Further information about topologies and their metrics can be found in [9].

For example, the work [24] presents an overview of four basic multi-agent topologic models:

1. a Web-like topology,
2. a Star-like topology,
3. a Grid-like topology, and
4. a Hierarchical Collective topology,

assesses their advantages and disadvantages in terms of agent autonomy, adaptation, scalability, and efficiency of cooperation.

The rest of this section will characterize the basic topologies utilized in all network structures: full topology, star topology, ring topology, tree topology and random topology.

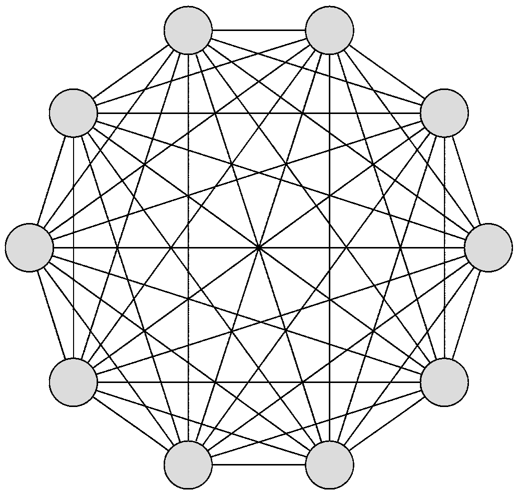
Fig. 1. Full topology

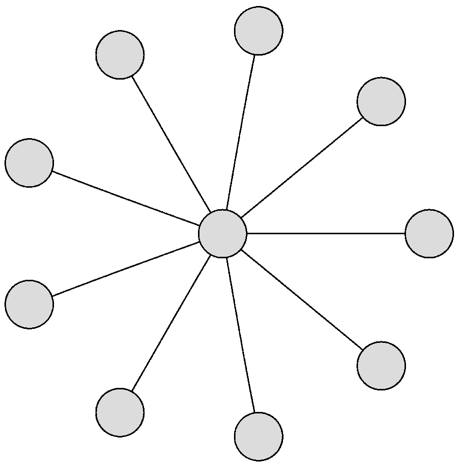
Fig. 2. Star topology

Figure 1 presents a distributed information system having a full topology. In this topology all entities communicate with each other directly. Thus there are no selected entities. Full topology is characterized by maximal level of redundancy;

but the direct communication between entities gives also the maximum reliability of system's operation, because the more entities a message has to cross on its way from sender to receiver, the more probable a failure. On the other hand, the redundancy present in fully connected topology introduces a huge overhead in entities' operation. The overhead is caused by excessive communication done by each entity. Full topology can be used in distributed systems built on networks offering high bandwidth, such as local area networks. The application of this topology in an environment with lower communication capacity can seriously harm system's performance.

Figure 2 presents a distributed information system having a star topology. In this topology there are two kinds of entities: one central entity and many peripheral entities. Peripheral entities connect only to the central one. Thus the central entity is connected to every other entity in the system. There is no redundancy in star topology. Therefore a failure of any of the links causes the system to be not connected. If one peripheral entity fails, the rest of the system can operate normally. The worst case scenario is when the central entity fails. In this case the system cannot operate at all. One can notice that the central element is the bottleneck of the system, because it is responsible for maintaining whole communication inside the system. The communication load on the central entity is also very burdening to it. It is worth noticing that the load of the central entity in this topology is the same as the load of every entity of a full topology.

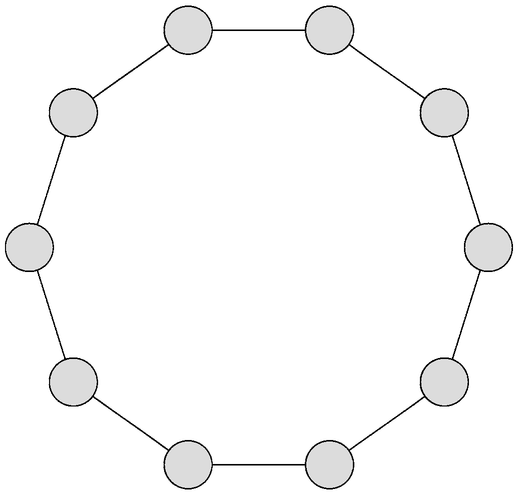
Fig. 3. Ring topology

Figure 3 presents a distributed information system having a ring topology. In this topology every entity has exactly two neighbors. Thus, the system is organized in one cycle containing all elements. Obviously, the redundancy is present in this topology. Between every two entities there are always two possible paths. The advantage of this topology is the increased reliability with a minimal cost. The

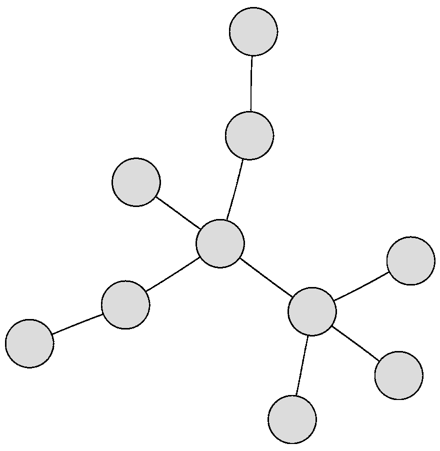
Fig. 4. Tree topology

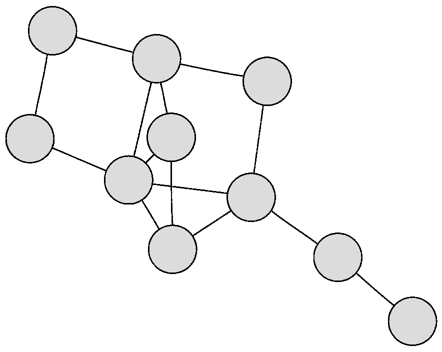
Fig. 5. Random topology

disadvantage of it is the long average distance between entities. It indicates that a message passed from one entity to another has to cross many indirect entities. Therefore, a probability of a failure is significant. On the other hand, the communication overhead introduced at each entity is minimal. Ring topology can be utilized in distributed systems not requiring fast data broadcasting inside a structure.

Figure 4 presents a distributed information system having a tree topology. In this topology there are no cycles. The redundancy is also not present as each pair of entities is connected by exactly one path. The advantage of this topology is that the communication overhead is balanced between all entities. The disadvantage is that the entities are equally important in the system's operation. This indicates that failure of any entity can seriously damage the whole system, because every entity is indispensable for communication. Tree topology can be utilized in distributed systems where a natural hierarchy is present. For example, when several academic centers build a common distributed system the tree topology would be a natural choice.

Figure 5 presents a distributed information system having a random topology. This topology is built in a random manner. There is no regularity or hierarchy in connections. The presence of a redundancy is therefore random. Also, the connections are not evenly distributed. There can be parts of the system which are highly interconnected and parts loosely interconnected. The general properties of such structures are difficult to assess. This topology can be utilized in a peer-to-peer system, where entities connect to each other spontaneously.

Apart from hardware configuration (i.e. computers and network) a special controlling software is needed to construct a distributed system. The main role of the controlling software is the coordination of the computers. The coordination consists of enabling communication, tasks dispensing and resources sharing. The communication is realized using a special protocol. The protocol must take into account the heterogeneity of the distributed system elements. Therefore the XML (eXtensible Markup Language) is often utilized. Furthermore, the design of the protocol must be fault tolerant to enable reliable communication. Tasks dispensing consists in distributing the elementary pieces of structure's goal among its elements. Task dispensing is vital to the distributed system's performance. It must take into account many factors, such as computers' parameters, their reliability, availability and current load. Resource sharing is designed for making the resources of one machine available to another. A resource can be both hardware and software. A hardware resource might be a special device or sufficient disk space. By the software resource usually a specific service is defined.

## 3.2 Errors in Information Systems

Exact examination of the sources of errors in distributed information systems is beyond the scope of this work. An error is considered as an improper state of one or more elements of a distributed system. This definition does not say anything about the nature of the error or its genesis. For the needs of studying error propagation phenomenon classification of the symptoms denoting the presence of the error seems to be more useful than classification of the errors themselves.

After Santoro [19], one can specify the following types of error symptoms occurring in distributed information systems: contamination, commission, omission, and crash.

The contamination error symptom manifests itself by invalid (not consistent with specification) data sent from one processor to another. The source of this symptom might be a fault application installed on the processor sending a contaminated data.

The commission error symptom is manifested by invalid communication behaviour consisting in sending a message not meant to be sent according to the protocol. It can be a result of a deliberate attack or of a software fault.

The omission error symptom manifests itself by not sending a message which should have been sent according to protocol. It can be a result of connection error.

The crash error symptom is caused by failure of the processor and consists in stopping its activity both connected to processing and communication.

The error propagation in this depiction is the series of *error* $\rightarrow$ *symptom* $\rightarrow$ *error* $\rightarrow \ldots \rightarrow$ *error* events. The error occurring in one element of the system causes the specific symptom to emerge and can cause another error in the other part of the system. It is important to notice that the error propagates itself along with the communication. The exception is the error causing the processor to crash. In this case the propagation consists in disability of system's operation. The processors depending on the crashed one cannot operate properly without it.

## 4 EXPERIMENTS

### 4.1 Model

The model of network system utilized in experiments on error propagation phenomenon consists of logical processes communicating with each other using the specified protocol. Each process models an operation of a single entity of a distributed system. Communication between processes takes place by means of messages exchange. The schema of the example model is presented in Figure 6.

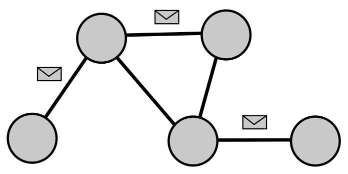

Fig. 6. Distributed system's model schema

The model assumes that distributed system is asynchronous. In practice this assumption means that the entities do not synchronize their actions with each other. Every entity works in real time with the native speed of its internal clock. The synchrony assumption makes implementing designing distribute algorithms much easier but in practice it is difficult to achieve. The second assumption states that the entities of the system are connected by bidirectional links. It is the common case in real-world systems. However, one must take into account that communication in TCP/IP network is bidirectional indeed but sometimes the establishment of connection can be initiated only by one of the entities involved. The simulation assumes that the communication channel can be initiated by both ends. The third assumption states that the entities are homogeneous – there is no selected entity fulfilling a special role. In reality there often exist one or more selected entities providing special services to the other entities. The best example is the Broker node of grid computing systems. The fourth assumption states that the entities know their neighbors and each entity is distinguishable (has unique id). In practice, the knowledge of the neighbors is not necessary for the entity to operate. Usually the elements of distributed systems know only one selected entity, which can provide information about the rest of the structure. The distinguish ability assumption is generally true in real-world systems. For instance, the IP number can be used as the ID of the entity. The last assumption states that the connection graph of the distributed system is connected. In practice it means that there is always a possibility of exchanging messages between any two entities being parts of the system.

Basically, the utilized model consists of entities $e \in \mathcal{E}$, where $\mathcal{E}$ is the set of all entities. The topology is defined by the binary neighborhood relation $\mathcal{N} \subset \mathcal{E} \times \mathcal{E}$ having the following properties:

**Symmetry** – $(e_i, e_j) \in \mathcal{N} \iff (e_j, e_i) \in \mathcal{N}$
**Aliorelativity (Irreflexivity)** – $(e_i, e_i) \notin \mathcal{N}$.

The first property states that if the entity $e_i$ is the neighbor of the entity $e_j$ then the entity $e_j$ must be a neighbor of $e_i$ also. This is the formal notation of the fact that bidirectional links are utilized in the model. The second property states that the entity cannot be a neighbor of itself. Using the neighborhood relation provided the set of neighbors of a specified entity can be defined as $n(e_i) = \{e_j \in \mathcal{E} : (e_i, e_j) \in \mathcal{N}\}$.

## 4.2 Algorithm

The error propagation can be observed as a progressing disturbance in designed system's operation. Thus modeling the system's operation is necessary to model the effects of this phenomenon and to trace its course. Therefore a simple distributed algorithm was designed and implemented. The algorithm was executed by the processes during the simulation.

To model distributed system's operation a distributed sum algorithm was implemented. The considered distributed sum problem is stated as follows: Every entity $e_i$ is given an initial random value $k(e_i)$ known only to this entity. During the execution of the algorithm, entities communicate with themselves exchanging information. After finite execution time each of the entities should know the exact sum of all initial values $S = \sum_{e \in \mathcal{E}} k(e_i)$.

The foregoing problem has some interesting features from the perspective of error research. Firstly, it requires intensive communication. The input information from one entity needs to be propagated to every other entity. Secondly the state of every entity depends on the state of the rest. The internal sum held by entity contains the cumulative information about the whole system. Thus, the distributed system solving this problem is sensitive to possible errors.

The solution involves spreading the knowledge across the distributed system and adjusting entities' internal states. The analysis of these problems provides very important information about the designed model. According to [19] it is impossible to design the protocol solving the distributed consensus problem in asynchronous systems in presence of faults without additional assumptions. The difficulty lies in the possible nontermination of every protocol for this problem. The distributed sum problem is the extended version of the consensus problem, because it requires the autonomic decision of every entity based on a common knowledge. Therefore constructing a model of asynchronous distributed system solving it in presence of faults is also impossible. Utilizing such model in simulation of error propagation phenomenon requires additional termination assumptions.

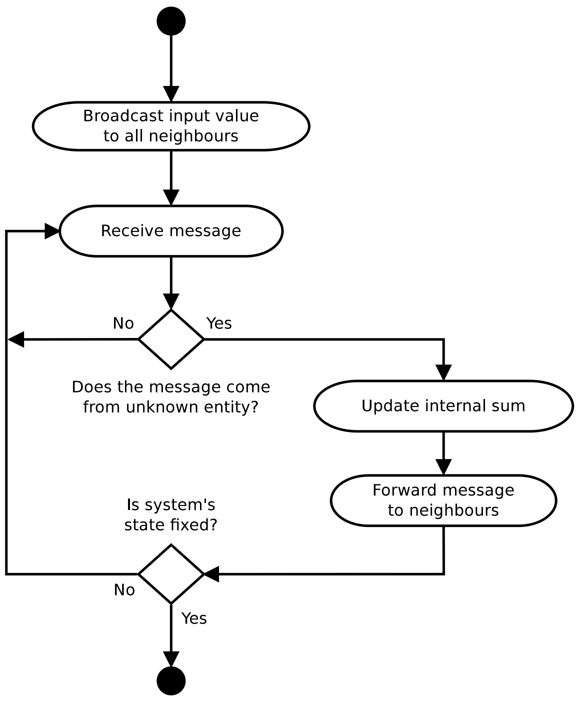
Broadcast input value to all neighbours
Receive message
No
Yes
Does the message come from unknown entity?
Update internal sum
Forward message to neighbours
Is system's state fixed?
No
Yes

Fig. 7. Distributed sum algorithm

The algorithm solving distributed sum problem is executed in the same way on every entity of the distributed system. The course of the algorithm on a single entity is presented in Figure 7 and Algorithm 1. The algorithm consists of a few steps. In the first step (line 1) the entity $e_i$ broadcasts its input value $k(e_i)$ to all of its neighbors. Then it waits for messages from the other entities (line 3). Incoming message is processed only if previously there were no other messages coming from this sender (line 4). It means that only the first message which was sent from given entity is taken into account. Message processing consists in increasing internal sum $o(e_i)$ by the value received and broadcasting the received value (lines 5 and 6). The process of receiving and broadcasting messages never terminates because the entity without knowledge of the system's structure cannot tell if it received messages from all of the other entities. However, the algorithm is designed in such way that when the entity does not change its internal sum when it receives a message from the entity from which it has received a message before. Thus, the state of the entity after receiving messages from all possible senders cannot change. Knowing that, the simulation system is able to terminate the algorithm execution after all entities have fixed their states.

```
broadcast input to all neighbours;
while not finished do
    receive message;
    if no previous messages from sender then
        update internal preference;
        forward message to all neighbors;
    else
        discard message;
    end
end
```

**Algorithm 1:** Distributed sum algorithm

The basic functioning of the distributed sum algorithm is the same as Flooding algorithm. There are four basic assumptions that need to be fulfilled so that Flooding works correctly. Connectivity requirement ensures that there exists a path between every two entities of the system. Without this assumption broadcasting would be impossible. Bidirectional links requirement states that the connection graph of the system should be undirected. This assumption prevents the situation when the source of the information cannot pass it to any other entity because it is only the destination of connection links. Unique initiator requirement states that the initiator of the broadcasting has to exist and it is the only entity that has the information before the algorithm starts. Finally the Total Reliability requirement states that the system's operation has to be failure-free. This assumption ensures that the broadcasting is not disturbed by failure of any entity.

As mentioned before, the distributed sum algorithm is based on Flooding algorithm, thus the requirements of the former are the superset of the requirements of the latter. The established model meets all of the Flooding algorithm's requirements explicitly, except the Total Reliability. In fact, the algorithm was designed to operate in a failure-free environment and the experiments examined its behaviour in case of different failures occurring in the system.

## 4.3 Introduced Errors

In order to examine the error propagation phenomenon in the described distributed system, fake error sources were introduced. The modified version of the distributed sum algorithm, with error sources introduced, is presented in Figure 8. The basic four types of errors were inserted into the model: Omission, Commission, Contamination and Crash. In the considered model, the Omission error consisted in the faulty entity not sending a message it should send. Conversely, the Commission error consisted in the faulty entity sending a message of a random content that should never been sent. The Contamination error resulted in random modification of the content of one of the sent messages. Finally, the Crash error caused the faulty entity to stop operating at all.

A single experiment consisted in inserting one of the error types specified above during the execution of the distributed sum algorithm. The error is introduced only once in the algorithm's operation on the exactly one selected entity of the system. Different errors occur in different algorithm steps. Algorithm 2 presents the distributed sum algorithm with error sources introduced. The Crash error can occur before the faulty entity receives a message (lines 3–5). If this happens, the entity immediately finishes all of its activities. The Omission error happens during the message broadcasting phase (lines 10–12). If it occurs, the entity does not send the message to one of its neighbors. The Commission error takes place after all of the messages have been sent (lines 21–23). If it occurs, the faulty entity creates a random message and sends it to one of its neighbors. The Contamination error causes the faulty entity to randomly modify one message sent to one of its neighbors during the broadcasting step (lines 13–15).

```
broadcast input to all neighbours;
while not finished do
    if crash error occurred then
        stop;
    end
    receive message;
    if no previous messages from sender then
        update internal preference;
        foreach neighbor in neighbors set do
            if omission error occurred then
                continue without sending message;
            end
            if contamination error occurred then
                change message randomly;
            end
            send message to neighbor;
        end
    else
        discard message;
    end
    if commission error occurred then
        send random message to randomly chosen neighbor;
    end
end
```

**Algorithm 2:** Distributed sum algorithm with error sources

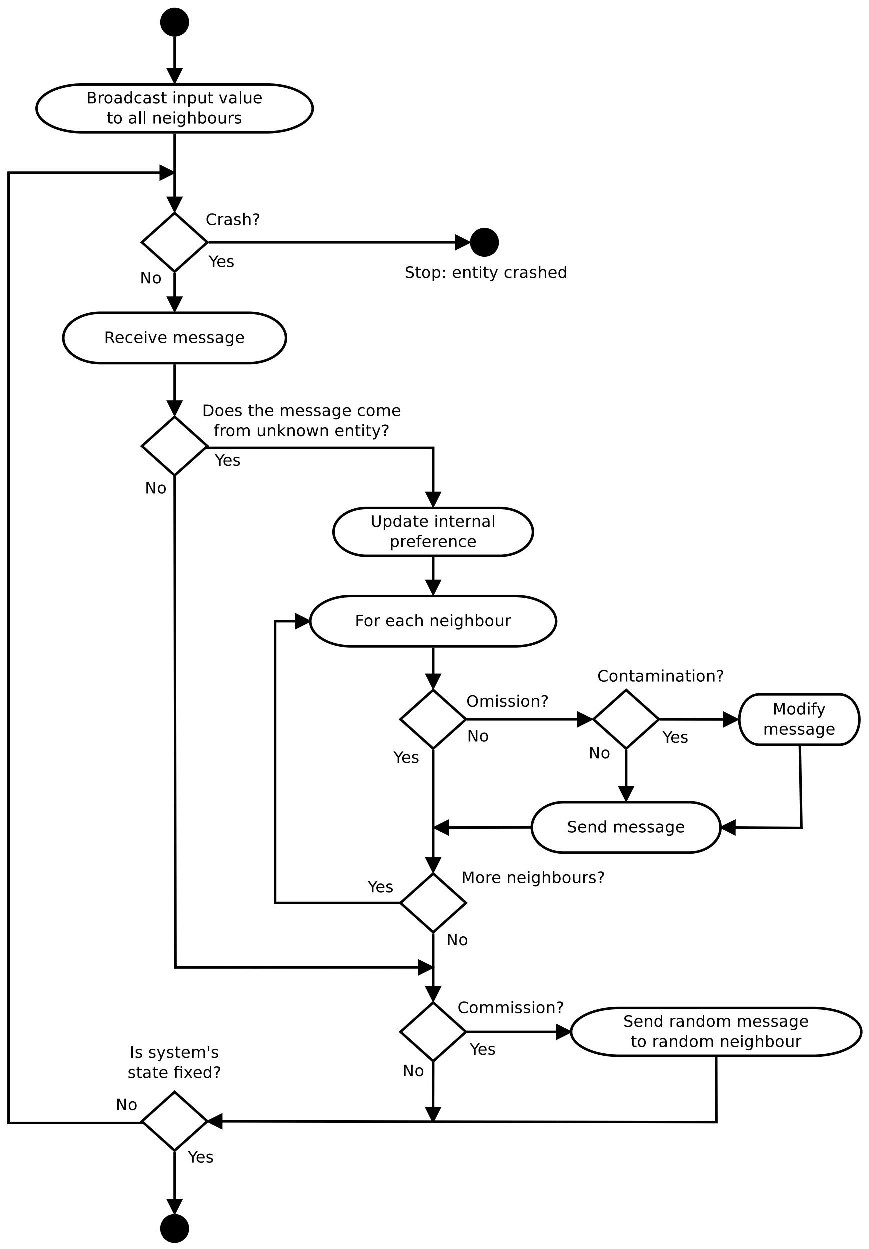

Fig. 8. Distributed sum algorithm with error sources

In order to check the influence of each error on the system at different points of time, the errors occur with come probability $p_e$. Introducing uncertainty in the model is necessary to verify, if the damage caused by the error is dependent on the moment of time when the error occurs.

The damage assessment was done by using the fraction of the entities' number affected by the error (having the internal state at the end of execution different than the $\mathcal{S}$ value). The *Error Percentage* value can be defined as $E_p = \frac{|F|}{|\mathcal{E}|}$, where $F = \{e \in \mathcal{E} : o(e) \neq \mathcal{S}\}$ is the set of error-affected entities. Also the algorithm operation time was used as a measure, because some errors did not directly affect the system's state but they did lengthened the execution time.

The *Error Percentage* value is the quantitative measure of the fault propagation phenomenon. It says nothing about the course of the propagation process but allows measuring its effects. Using this measure makes the statistical analysis of the experiment results possible.

## 4.4 Testbed and Implementation Details

The simulation environment was constructed using Python [17] programming language. The most recent, stable version (2.5) of Python was used. This language is being developed by *Python Software Foundation.* The choice of this programming tool is justified by several reasons. Python programming language is mature and commonly used technology. Its usage is free even in commercial applications. Furthermore, Python community offers outspreaded support for the programmers. Another advantage of this language is the great number of professional third party libraries extending its capabilities. Many of the libraries are designed for scientific applications. This work utilized the SimPy library.

"SimPy (Simulation in Python) is an object-oriented, process-based discrete-event simulation language based on standard Python" [20]. It is being continuously developed since 2004 by *SimPy Developer Team.* SimPy provides powerful modeling features and provides a simple and elegant interface. Moreover, it supplies programmers with advanced data collection and visualization tools. It has been successfully applied in such domains as epidemics simulation, traffic simulation, industrial engineering and more.

The conducted experiments were the source of large amounts of data. Employing the database system was necessary for the analysis of the results. The *Postgre SQL 8.1* database was utilized in order to complete this task. *Postgre SQL* is one of the most advanced open-source database systems. It stands up to commercial solutions in the domain of performance and tools available. In order to utilize this database system with the Python language the PyGreSQL library was used.

For statistical analysis the *R* programming language was used. This language is an open-source version of the *S* language developed at Bell Laboratories. Initially the *R* language project was developed at the Statistic Department of the University of Auckland. The *R* language offers a wide range of statistical and graphical techniques. It was utilized for data processing and visualization.

Another tool utilized in this work was the *Graphviz* graph visualization tool. Most parts of this application is being developed by AT & T Research. This tool was used for the visual analysis of the fault propagation phenomenon.

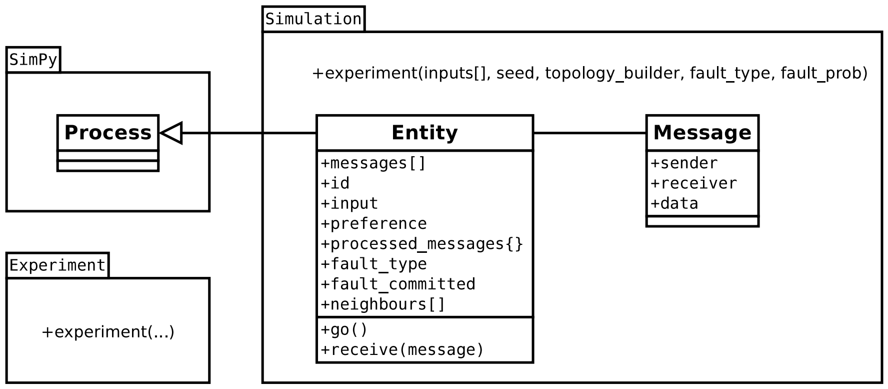

Fig. 9. Class diagram of the simulation system

Figure 9 presents the UML class diagram of the implemented simulation system. The most important element of the application is the *Entity* class. It models the state and behaviour of a single entity of distributed information system. It is inherited from the SimPy's *Process* class. The state variables include the *messages* list, the *id* variable, the *input* variable, the *preference* variable, the *processed_messages* set and the two fault controlling variables – *fault_type* and *fault_committed*. The *messages* list is a buffer of messages that were received by the entity but not yet processed. They are ordered by the time they came in. The *id* variable contains the unique identifier of the entity. No two entities have the same *id* value. The *input* variable contains the value of input of the entity. It is generated randomly at the *Entity* object initialization. The *preference* variable contains the value of the distributed sum computed at the entity. Initially it is set to the value of the *input* variable and is modified by the course to the algorithm. The *processed_messages* variable stores a set of the messages that have been already processed by the entity. Its main purpose is to keep track of the information that the entity has. The *processed_messages* set is utilized for filtering incoming messages – those coming from the already known entities are not processed again.The *fault_type* variable can adopt one of the five values: *None*, *Omission*, *Commission*, *Contamination* and *Crash*. It controls the faulty behaviour of the entity. The *fault_committed* variable contains a logical value indicating whether the entity had already committed a fault or not. Holding this value is necessary, because according to the model's assumptions the fault should occur only once during the simulation.

The operation of the entity is modeled inside the *go* method. This method implements the behaviour presented in Figure 8. Some language specific modifications to the algorithm have been made in order to optimize the method's operation. Additionally, after every operation performed by the entity, delaying instructions have been inserted to emulate the execution time of this operation.

The *receive* helper method is used when the entity receives a message. It simply appends the incoming message to the entity's messages queue.

The *Message* holds the information about a single message exchanged by the pair of entities. It has three fields: *sender*, *receiver* and *data*.

Conducting experiment consists in creating the set of entities, assigning input values to them, connecting them into the specified topology, choosing a faulty entity and a fault type and starting the simulation. These activities are executed by the *experiment* global method. The method accepts the following parameters: *inputs*, *seed*, *topology_builder*, *fault_type*, *fault_prob*. The *inputs* parameter contains the array of the input values to be assigned to the entities. The *seed* parameter is the random seed, which is set before each simulation. Random seed is utilized by the pseudorandom number generator to construct the random values. The seed needs to be assigned, because the simulation bases on some random events (such as error occurrence or entities actions' duration). The *topology_builder* parameter stores the function which accepts the array of entities and connects them into the specified topology. The *fault_type* and *fault_prob* parameters store the information about the type of the error that will occur in the simulation and its probability accordingly.

For the results of the experiments to be statistically significant, it is necessary to execute multiple simulations with varying parameters. The *xperiment* package has been designed and implemented to complete this task. Basically, the tools included in this package allow executing the experiments automatically and collecting the results. The main method of the package is the *xperiment.experiment* method. In order to conduct the set of experiments, one should pass the method under examination and its parameters space to the *xperiment.experiment* method. The *xperiment.experiment* method executes the method passed to it with every combination of the input parameters specified. The *xperiment* package defines a set of logging classes. These classes allow writing the results of the experiments to the console, file or database table.

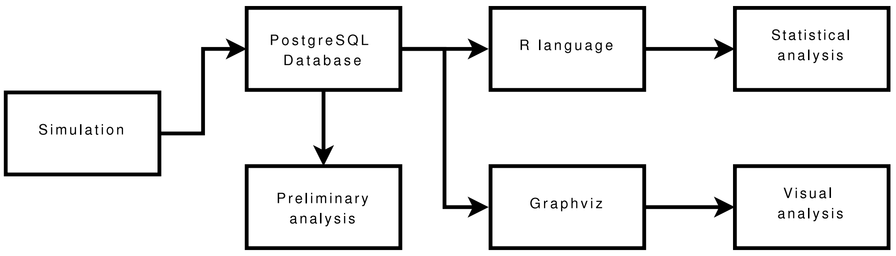

Fig. 10. Results processing schema

Figure 10 presents the schema for results processing. The results collected during simulation consist of the simulation parameters and the measures calculated based on the experiment progression. Simulation parameters include: utilized topology, the number of entities, error probability and error type. The measures are error percentage, the graph representing system's state at the end of simulation and simulation time. Every simulation generates one set of results which is saved to the database. The results are preliminarily checked for consistency and sanity inside the database environment using SQL queries. Then data is exported to a file and loaded into the R language environment. The statistical analysis and visualization is performed there. Additionally the state graphs of the selected cases are extracted from the database to files. Then they can be visualized using the Graphviz tool and analyzed.

## 5 RESULTS ANALYSIS

As the results of the experiments 6 000 observations were saved in the database. Each observation described a single simulation. The simulation is described by the number of entities of the simulated system (10, 20, 30, 40 50 and 100), the topology of the system (star, ring, full, tree and random), the probability of the error occurrence (0.1, 0.3, 0.5, 0.8, 0.9) and the type of the error (none, contamination, commission, omission and crash). The parameters mentioned above create the four-dimensional discrete experiment space. At each point of this space, the output values like error percentage or simulation time were measured. In order to get statistically significant results, each point of the experiment space was examined ten times. It was necessary to repeat the simulation many times, because the course of a simulation was not determined in advance. Each time the simulation was ran, the faulty entity was drawn. Furthermore, the exact moment of error occurrence was also drawn.

### 5.1 Quantitative Analysis

| | *Error Type* | | | | | |
|---|---|---|---|---|---|---|
| *Topology* | **None** | **Omission** | **Commission** | **Contam.** | **Crash** | **Avg.** |
| **Full** | 0 | 0 | 0 | 0.04 | 0.51 | 0.11 |
| **Random** | 0 | 0.05 | 0.16 | 0.28 | 0.55 | 0.21 |
| **Ring** | 0 | 0 | 0.39 | 0.44 | 0.51 | 0.27 |
| **Star** | 0 | 0.49 | 0.31 | 0.46 | 0.58 | 0.37 |
| **Tree** | 0 | 0.52 | 0.31 | 0.52 | 0.69 | 0.41 |
| **Avg.** | 0 | 0.21 | 0.23 | 0.35 | 0.57 | 0.27 |

Table 1. Average error percentage relationship to topology and error type

Table 1 presents the average error percentage relationship to the topology and error type. A single row contains the average error percentage in the selected topology with different error types occurred. The last column is the average error percentage of the topology in all experiments. A single column contains the average error percentage caused by the selected error type in different topologies. The last row is the average error percentage in all topologies. The rows are ordered from the topology least susceptible to errors to the most susceptible one. The columns are ordered from the least malicious error type to the most malicious one.

The quantitative analysis is not very accurate, because it simplifies the results. Nevertheless, it leads to some important conclusions. Firstly, the model's implementation is correct, because all simulations ran without error introduced (error type is None) completed with zero error percentage. It implies that the errors occurred during simulation are the result of an intendant error introduction. Secondly, topologies containing cycles (Full, Random and Ring) seem to be much more immune to errors than the ones without cycles (Star and Tree). A cycle in a topology is a form of redundancy. As mentioned before, the redundancy is the basic mean of implementing fault-tolerant systems. The designed models did not follow the fault-tolerance guidelines explicitly. Nevertheless, the algorithm of calculating a distributed sum (see Section 4.2) appears to be designed in a way taking advantage of the topology redundancy. Further analysis of this phenomenon will be conducted in the Visual analysis (Section 5.2). Thirdly, topologies behave differently in case of different error types. The general harmfulness ordering of error types is almost the same in all topologies. The exceptions are the Tree and Star topologies, where the Commission errors seem to be less harmful than Omission errors. The exact values of error percentage of each error type differ depending on the type of topology. Fourthly, the topologies containing cycles are much less susceptible to omission errors than the ones not containing cycles. The reason for that fact could be that the redundancy present in these topologies cancels the information omission.

| | *Topology* | | | | | |
|---|---|---|---|---|---|---|
| *Entities Count* | **Full** | **Random** | **Ring** | **Central** | **Tree** | **Avg.** |
| **10** | 0.13 | 0.21 | 0.23 | 0.31 | 0.4 | 0.26 |
| **20** | 0.09 | 0.2 | 0.26 | 0.35 | 0.39 | 0.26 |
| **30** | 0.13 | 0.2 | 0.27 | 0.34 | 0.44 | 0.28 |
| **40** | 0.11 | 0.18 | 0.27 | 0.43 | 0.41 | 0.28 |
| **50** | 0.09 | 0.24 | 0.28 | 0.39 | 0.44 | 0.29 |
| **100** | 0.11 | 0.22 | 0.29 | 0.39 | 0.39 | 0.28 |
| **Avg.** | 0.11 | 0.21 | 0.27 | 0.37 | 0.41 | 0.27 |
| **Standard Deviation** | 0.02 | 0.02 | 0.02 | 0.04 | 0.02 | 0.01 |

Table 2. Average error percentage relationship to topology and entities count

Table 2 presents the relationship between the average error percentage and topology and entities count. The columns of the table contain different topologies types. The rows of the table contains the different number of entities of the simulated distributed system. The study of Table 2 provides one important conclusion. The error percentage measured for the same topology but for different number of entities seem to be constant. The conclusion might be that the number of entities does not influence the error percentage. The reason for that could be the error percentage measure is the fraction of the number of entities. Nevertheless, this measure should reveal the possible effect of scale of error propagation phenomenon.

| | *Topology* | | | | | |
|---|---|---|---|---|---|---|
| *Error probability* | **Full** | **Random** | **Ring** | **Central** | **Tree** | **Avg.** |
| **0.1** | 0.03 | 0.07 | 0.12 | 0.07 | 0.15 | 0.09 |
| **0.3** | 0.06 | 0.15 | 0.21 | 0.23 | 0.33 | 0.2 |
| **0.5** | 0.11 | 0.23 | 0.28 | 0.38 | 0.42 | 0.28 |
| **0.8** | 0.17 | 0.28 | 0.34 | 0.53 | 0.53 | 0.37 |
| **0.9** | 0.17 | 0.3 | 0.36 | 0.61 | 0.58 | 0.4 |

Table 3. Average error percentage relationship to topology and error probability

Table 3 presents the relationship between the average error percentage and topology and error probability. The columns of the table contain different topology types. The rows of the table contain different error probabilities utilized in simulations. As mentioned in Section 4.3, the role of error probability parameter is to differentiate the error occurrences in time. In the model of distributed systems, the error probability parameter is used by the function determining the moment of entity's error. From the statistical point of view, the higher the probability the sooner an error occurs. The analysis of Table 3 leads to the following conclusion: there seems to be a directly proportional relation between the error probability and error percentage. This regularity can be observed in every topology. One of the possible explanations for that fact could be that the distributed sum algorithm is much more susceptible to errors occurring in the early phase of its execution. As mentioned before, the higher the probability the sooner the error.

| | *Topology* | | | | |
|---|---|---|---|---|---|
| *Error Type* | **Full** | **Random** | **Ring** | **Star** | **Tree** |
| **None** | 7924.93 | 2863.42 | 782.53 | 7798.76 | 1442.9 |
| **Omission** | 11924.21 | 91546.69 | 900.44 | 1256760.08 | 1684114.09 |
| **Commission** | 7921.92 | 2859.37 | 784.3 | 7793.62 | 1446.45 |
| **Contamination** | 7922.33 | 2853.87 | 781.79 | 7797.24 | 1445.67 |
| **Crash** | 1168883.85 | 1335311.8 | 1204626.81 | 1421463.51 | 2054520.83 |

Table 4. Average simulation time relationship to topology and error probability

Table 4 presents the relationship between the average simulation time and the topology and error probability. The columns of the table contain different topology types. The rows of the table contain different error types occurred during simulations. The analysis of the average simulation time seems to confirm the hypothesis. The occurrence of errors consisting in data loss (Omission, Crash) indeed lengthened the simulation time significantly. Even the topologies perfectly immune to omission error (ring, full) operated slower in presence of this kind of error. Separate explanation needs to be made in case of crash error. The simulation was recognized as complete in the event of fixed structure's state affirmation or the exceedence of simulation timeout limit arbitrarily set to 100 000 time units. The latter condition ensured that the simulation would eventually finish. The timeout limit was set to a value hardly probable in normal algorithm execution. So, when the simulation execution time is equal to the timeout value it means that the structure's operation was disturbed in a way making the fixed state stop condition impossible to meet. Thus, the average simulation time a few times longer than the normal indicates that there was at least one simulation exceeding the timeout. This happens when the faulty entity did not manage to broadcast its initial value before it has crashed. So the value of average simulation time is not reliable in this case. Another observation is that the other kinds of errors (commission and contamination) do not influence the average simulation time significantly.

## 5.2 Qualitative Analysis

The error percentage measure is a purely quantitive observation. It says nothing about the course of the error propagation phenomenon. Visual analysis was conducted in order to fill this gap. The source of data utilized in this section is the snapshot of the model's state at the end of simulation. This snapshot takes the form of graph in the *dot* file format saved in the database along with the other simulation results. The *dot* file format can be visualized using the *Graphviz* software and then analyzed.

The qualitative analysis consisted of examining the snapshot of the final state of the model at the end of simulation and trying to determine the error propagation pattern. This method was meant to provide the quantitive observations to explain the process of error propagation. The disadvantage of visual analysis is the lack of automation and thereby the inability of processing the whole range of results. Additionally, analysis of the larger structures consisting of tens of entities is very laborious. Therefore only some of the simulations were chosen for the analysis. Nevertheless, visual analysis is a very important part of the results processing.

The snapshots of the systems were visualized and presented as figures. The following conventions were used during the visualizations:

- the snapshot is visualized as the undirected graph,
- each entity is presented as a vertex and each connection is represented by the edge,
- the numbers inside the vertexes indicate their id,
- the faulty entity is marked with dark grey color,
- the entities affected by the error (having the invalid state at the end of simulation) are marked with the light grey color,
- the entities unaffected by the error are marked with the white color.

In order to compare the error propagation in different topologies one might want to group the results of the analysis by the error type; but in a single topology the error propagation phenomenon proceeds similarly in spite of the error type. Thus, the results are grouped by the topology to show the differences in the topology behaviour in case of different errors.

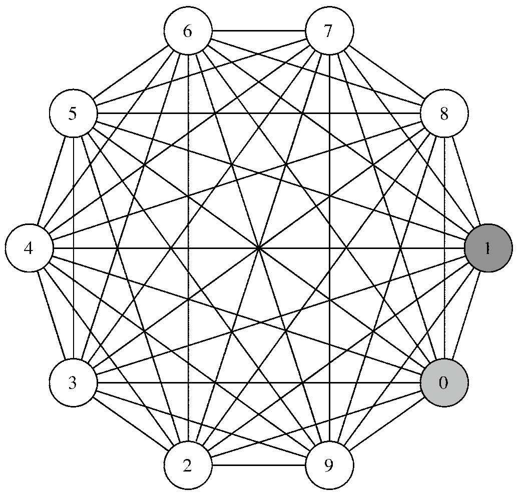

Fig. 11. Visualization of contamination error propagation in full topology

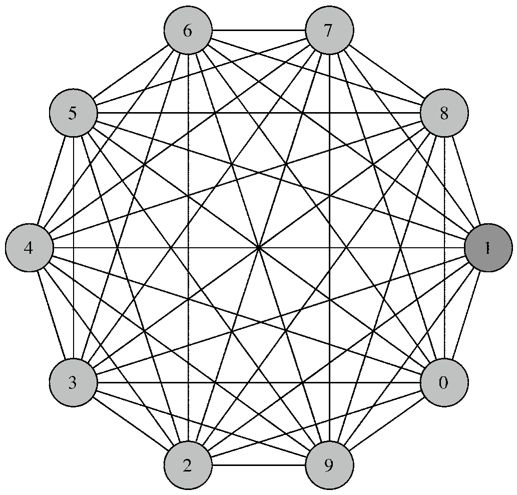

Fig. 12. Visualization of crash error propagation in full topology

Figure 11 presents the visualization of the contamination error propagation in full topology. *Entity 1* is the source of the error. *Entity 0* is affected by the error. The propagation path in this case is simple. *Entity 1* probably contaminated a message to *Entity 0* during the broadcasting phase. The other scenario that the contaminated message came from another entity is hardly probable because the error could only occur after *Entity 1* had finished broadcasting its initial value and by this time *Entity 0* would probably get the proper value from the original source of the message. It is important to notice that even though *Entity 0* is affected by the contamination, the error does not propagate any further. The other entities (including *Entity 1*) hold the valid sum at the end of algorithm's execution. This is because of the redundancy present in full topology. *Entity 0* cannot contaminate other entities, because they already received the original value from its source.

Figure 12 presents the visualization of the crash error propagation in full topology. One of two possible scenarios is presented. In the presented case, *Entity 1* crashed before it managed to pass its input value to any other entity. This resulted in the invalid final state of the whole system. It is worth noticing that if *Entity 1* had sent a message containing its initial value to even one entity, the receiver would successfully broadcast it to others. This is the second possible scenario, when the only entity affected by the crash is *Entity 1* itself. The situation when the crashed entity still has the correct state at the end of the simulation is also probable. In this case, the crash should occur after the faulty entity had broadcasted its initial value and collected the values from the other entities. The described scenario did not occur in experiments.

The other error types (commission and omission) are not analysed, because the full topology seems to be immune to these kind of errors. The commission error does not influence the full topology, because by the time the false message is delivered to the receiver, the receiver already has the real one. The omission error is cancelled because even if one of the entities does not receive the message directly from its source it still receives a copy from the other entity. It is clear that the redundancy of the full topology makes it almost completely fault-tolerant.

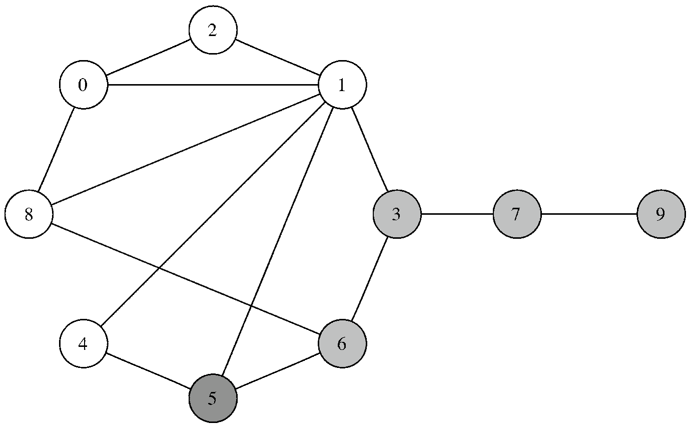

Fig. 13. Visualization of commission error propagation in random topology

Figure 13 presents the visualization of the commission error propagation in random topology. *Entity 5* is the source of the error and *Entities 6, 3, 7, 9* are affected

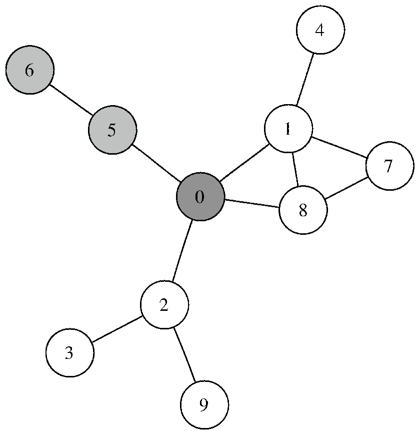

Fig. 14. Visualization of omission error propagation in random topology

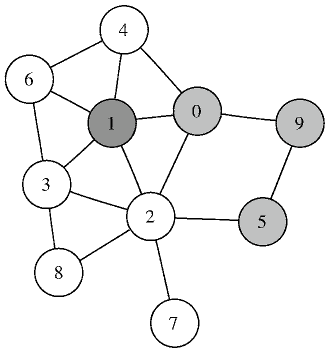

Fig. 15. Visualization of contamination error propagation in random topology

by the error. The error propagation phenomenon in this case proceeded in the following way: *Entity 5* sent a false message to *Entity 6*. As a sender of this message one of the entities unaffected by error was drawn. A false message had to be received at *Entity 6* before the real message from sender was received. Therefore *Entities 0, 1, 2, 4* are the most probable senders as they are not directly connected with *Entity 6*. *Entity 6* propagates the false value to *Entity 3*, *Entity 7* and finally to *Entity 9*. *Entity 1* is not affected by error, because it had received the real message before *Entity 3* sent the false one to it. The important conclusion coming from the analysis of this case is that error propagation in utilized model depends highly on time.

Figure 14 presents the visualization of the omission error propagation in random topology. *Entity 0* is the source of the error and *Entities 5, 6* are affected by the error. *Entity 0* connects the affected entities with the rest of the structure. Therefore every message coming from the unaffected entities must pass through *Entity 0* to reach *Entities 5, 6*. In this case, *Entity 0* omitted a message which was meant to be sent to *Entity 5*. The lack of redundant connections to *Entities 5, 6* caused that they could not get the message from the other source and eventually resulted in their invalid final state.

Figure 15 presents visualization of the contamination error propagation in random topology. *Entity 1* is the source of the error and *Entities 0, 5, 9* are affected by the error. The course of the error propagation seems to be obvious. *Entity 1* contaminated a message to *Entity 0*, and this message was then passed further, to *Entity 9* and *Entity 5*. Again, the time factor seems to be deciding in this case. It is worth noticing that *Entity 2* received the proper value because it holds the right state at the end of simulation. Furthermore, it had to receive it before *Entity 0* passed the contaminated message to it. But still, the error propagation path (*Entity 0-¿Entity 9-¿Entity 5*) had to be shorter (in the time dimension) than the path from *Entity 2* to *Entity 5*. It could be caused by the fact that *Entity 2* has 6 neighbors and therefore processing and broadcasting each message takes much more time than with *Entity 9* having just 2 neighbors.

The course of the crash error propagation in random topology is in most cases the same as in full topology. However, there are rare situations, when the crashed entity cuts off some part of the structure. In this case, the separated group of entities holds the improper value at the end of simulation. In the other cases only the faulty entity is affected by the error or the whole structure fails. The exact scenario of the crash error propagation depends on the time when the error occurs.

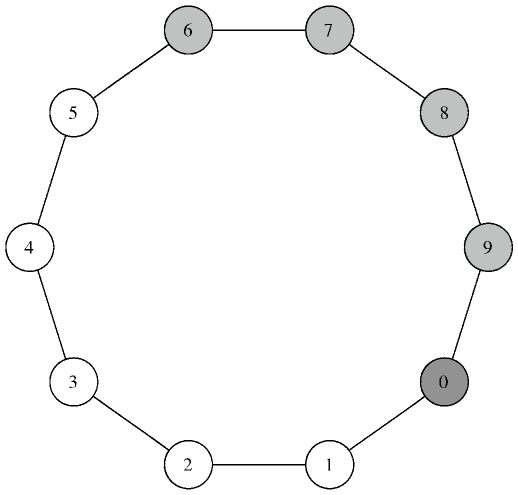

Fig. 16. Visualization of commission and contamination errors propagation in ring topology

Figure 16 presents the visualization of the commission and contamination errors propagation in ring topology. *Entity 0* is the source of the error and *Entities 6, 7, 8, 9* are affected by the error. The course of the propagation is obvious. *Entity 0* generates the error (contamination or commission) and then the error is propagated

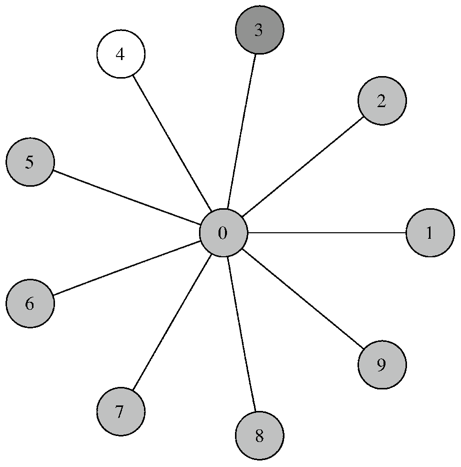

Fig. 17. Visualization of commission error propagation in star topology

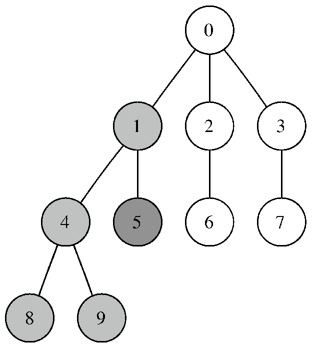

Fig. 18. Visualization of commission error propagation in tree topology

to the affected entities. The propagation finishes at *Entity 6*, because *Entity 5* had received the proper value before the invalid one has been passed to it.

Ring topology is immune to omission errors, because the loss of one message is always compensated by the alternative message delivery path. The course of the crash error propagation is similar to the ones in the two previously presented topologies.

Figure 17 presents the visualization of the commission error propagation in the star topology. *Entity 3* is the source of the error and all entities except for *Entity 4* are affected by the error. *Entity 3* committed a false message to the central entity *Entity 0*. *Entity 0* propagated the message to the rest of the entities. The only entity unaffected by the error is *Entity 4*. It was not influenced by error, because the false message had the sender field set to this entity, and therefore the false value was not taken into account.

The course of the error propagation phenomenon in star topology depends highly on the place of the error occurrence. Paradoxically, when the central entity fails the damage caused by the error is not large. The reason for that is the error occurred at the central entity propagates only to the one entity, because the peripheral entity being the receiver of the false message cannot propagate it any further. The other ones are not affected. The exception for this rule is the crash error of the central entity, because it results in impossibility of any communication between the other entities. In this case the whole structure is damaged. On the other hand, when the peripheral entity is faulty the error passed to the central entity is passed to the rest of the structure. Thus the whole structure becomes affected. This situation is presented in Figure 17. Once again, the exception is the case, when the peripheral entity crashes. In this case, the rest of the structure is operating normally.

Figure 18 presents the visualization of the commission error propagation in the topology. *Entity 5* is the source of the error and *Entities 1, 4, 8, 9* are affected by the error. The error is propagated from *Entity 5* to *Entity 1* and then down the hierarchy to *Entities 4, 8, 9.* Propagation up the hierarchy is stopped at *Entity 1* because the *Entity 0* received a proper value before.

The other kinds of errors propagate in similar way in tree topology. The lack of redundancy in tree topology causes the error can propagate freely across the structure. The damage caused by the error depends highly on the place of its occurrence. The higher in hierarchy the faulty node is the more damage the error causes.

## 6 CONCLUSIONS

It can be concluded from the experimental results and analysis that the course and range of the error propagation phenomenon are influenced by topology of the distributed information system.

The examination of the error propagation phenomenon by means of quantitative and qualitative analyses leads to the following conclusions. Firstly, the topologies utilized in network systems can be ordered from the most to the least immune to error propagation as follows:

1. full topology,
2. random topology,
3. ring topology,
4. star topology and
5. tree topology.

According to [5] findings that scale-free graphs make the system as efficient as fully connected graphs we understand how scale-free networks are particularly well immune to error propagation. Scale-free networks have a structure, which makes their performance less robust to agent breakdowns or network failures [7]. Studying methods for developing strategies, and responding to failures in the network, could potentially lead to mitigating the negative effects of certain interaction topologies and might allow to overcome deficits in individual connectivity. One possibility is to allow the entities to adapt network connectivity by adding and removing local connections. Other possibilities include collaboration. All of these organizational paradigms are subjects of our ongoing research. Another interesting finding is that the viruses and other threats can spread rapidly in small-world networks, for example the social network formed by SMS users [12], where the most common are star and tree topology.

Qualitative analysis provides detailed explanations for this state of affairs. Full topology is the most immune one, because it contains the most highly connected topology. It is the most redundant structure of the examined. Random topology performs slightly better than ring topology, probably because of the average distance between entities. Both topologies contain redundancy, but an error propagating in ring topology affects more entities, because the invalid message traverses the bigger part of the structure before it gets detected and corrected. The weakest immunity to error have the topologies not employing redundancy – the tree and star topology. And again the topology having lower average inter-entities distance (the star topology) performs slightly better.

Other observations concern error propagation of different error types. The error types can be ordered form the least to the most harmful:

1. omission error,
2. commission error,
3. contamination error,
4. crash error.

The omission error can be easily cancelled by the redundancy. The results presented in Table 1 indicate that the topologies employing redundancy (the full, random and ring topologies) are almost completely immune to this kind of error. Visual analysis pointed out that the information lost due to the omission error is cancelled by the presence of the replicated data in the other entities. On the other hand, when the redundancy is not present (like in star and tree topologies), the information is ultimately lost. The commission error is slightly less harmful than the contamination error. The nature of these kind of errors seems similar – the damage is propagated along with the modified message. The difference is that in commission error the message is created by one of the entities and in contamination error the existing message is modified. The modification is more harmful, because when the real message is affected, the situation when its receiver does not have the proper value already is more probable. Crash error is the most harmful error type, because it often leads to disconnection of the topology and irreparable information lost.

Examination of the influence of error probability on error propagation phenomenon leads to the conclusion that the sooner an error occurs the more damage it causes. It is illustrated in Table 3. This relationship can be explained by using visual analysis. When the error occurs in the early stage of distributed sum algorithm execution, the structure is in an empty state. Every value emerging within the system is received by entities and passed further. In later stages, the refusal of the improper message is more probable, because the entities mostly contain the right value already.

The main obstacle of this work was the complexity of the topic of analysis. In order to conduct experiments the simplified model of a distributed system was designed and implemented. But still, the calculations connected to simulations took a week of computing time to complete. This is the main reason, why the experiments concerned relatively small distributed systems – up to 100 entities. It might be a counter-argument challenging the hypothesis, but the obtained results have shown that the number of entities has little or no influence on the course of error propagation.

The study of error propagation phenomenon in distributed information systems has shown this issue is complex and worth further research. This work concerns only a small part of this area. Future works in this subject should analyse deeply the factors having the influence on the error propagation. The results presented denied the connectivity to be the factor influencing the course of error propagation, but other metrics of the topology can be specified and examined. In order to verify the hypothesis the error percentage measure was applied. As stated in Section 4 this measure is oversimplified. It is quantitative description of the effects of the phenomenon. It says nothing about the course of the propagation. Future work should introduce the qualitative measure of error propagation allowing analyzing its course. The presented results can be also analysed in a different way than in this work. For instance, the correlation between the different parameters of social systems and the error propagation measures could be calculated. Data mining techniques could also be utilized in order to discover the associations in the domain of error propagation not examined yet. Finally, the real-world system experiment should be conducted in order to verify the given hypothesis in practice.

## REFERENCES

[1] Albert, R.—Jeong, H.—Barabsi, A.-L.: Error and Attack Tolerance of Complex Networks. Nature, Vol. 406, 1999, pp. 378–381.

[2] Bales, M. E.—Johnson, S. B.: Graph Theoretic Modeling of Large-Scale Semantic Networks. Journal of Biomedical Informatics, Vol. 39, 2006, pp. 451–464.

[3] Cami, A.—Deo, N.: Techniques for Analyzing Dynamic Random Graph Models of Web-Like Networks: An overview. Networks, Vol. 51, 2008, No. 4, pp. 211–255.

[4] Chang, B. M.—Jo, J. W.—Her, S. H.: Visualization of Exception Propagation for Java Using Static Analysis. In: Proceedings of the Second IEEE International Workshop on Source Code Analysis and Manipulation, Washington, USA, 2002, pp. 173–182.

[5] Delgado, J.: Emergence of Social Conventions in Complex Networks. Artificial Intelligence, Vol. 141, 2002, pp. 171–185.

[6] Gabbar, H. A.: Improved Qualitative Fault Propagation Analysis. Journal of Loss Prevention in the Process Industries, Vol. 20, 2007, pp. 260–270.

[7] Gaston, M. E.—des Jardins, M.: The Effect of Network Structure on Dynamic Team Formation in Multi-Agent Systems. Computational Intelligence, Vol. 24, 2008, No. 2, pp. 122–157.

[8] Król, D.—Kukla, G.: Reusable Grid Computing Architecture Based on Code Propagation Method. Int. J. Intelligent Information and Database Systems, Vol. 3, 2009, No. 1, pp. 44–55.

[9] Król, D.—Zelmozer, M.: Structural Performance Evaluation of Multi-Agent Systems. Journal of Universal Computer Science, Vol. 14, 2008, No. 7, pp. 1154–1178.

[10] Lin, J.—Kuo, S.: Resolving Error Propagation in Distributed Systems. Information Processing Letters, Vol. 74, 2000, No. 5-6, pp. 257–262.

[11] Liu, L.—Antonopoulos, N.—Mackin, S.: Managing Peer-To-Peer Networks With Human Tactics in Social Interactions. J. Supercomput., Vol. 44, 2008, pp. 217–236.

[12] Meng, X.—Zerfos, P.—Samanta, V.—Wong, S. H. Y.—Lu, S.: Analysis of the Reliability of a Nationwide Short Message Service. In: Proceedings of 26th IEEE International Conference on Computer Communications, 2007, pp. 1811–1819.

[13] Milo, R.—Shen-Orr, S.—Itzkovitz, S.—Kashtan, N.—Chklovskii, D.—Alon, U.: Network Motifs: Simple Building Blocks of Complex Networks. Science, Vol. 298, 2002, pp. 824–827.

[14] Martel, M.: Semantics of Roundoff Error Propagation in Finite Precision Calculations. Higher-Order and Symbolic Computation, Vol. 19, 2006, No. 1, pp. 7–30.

[15] Nikoletseas, S.—Prasinos, G.—Spirakis, P.—Zaroliagis, C.: Attack Propagation in Networks. In: Proceedings of the 13th Annual ACM Symposium on Parallel Algorithms and Architectures, New York, USA, 2001, pp. 67–76.

[16] Newman, M.: The Structure and Function of Complex Networks. SIAM Review, Vol. 45, 2003, pp. 167–256.

[17] Python Development Team. Python Programming Language Home Page. Available on: `http://python.org`, 2008.

[18] Sakata, S.—Yamamori, T.: Topological Relationships Between Brain and Social Networks. Neural Networks, Vol. 20, 2007, pp. 12–21.

[19] Santoro, N.: Design and Analysis of Distributed Algorithms. John Wiley & Sons, 2007.

[20] SimPy Development Team. Simpy Discrete Event Simulation Framework Home Page. Available on: `http://simpy.sourceforge.net/`, 2008.

[21] Strogatz, S. H.: Exploring Complex Networks. Nature, Vol. 410, 2001, pp. 268–276.

[22] Thain, D.—Livny, M.: Error Scope on a Computational Grid: Theory and Practice. In: Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing, 2002, pp. 199–208.

[23] Watts, D. J.—Strogatz, S. H.: Collective Dynamics of "Small-World" Networks. Nature, Vol. 393, 1998, pp. 440–442.

[24] Zhu, Q.: Topologies of Agents Interactions in Knowledge Intensive Multi-Agent Systems for Networked Information Services. Advanced Engineering Informatics, Vol. 20, 2006, pp. 31–45.

**Dariusz Król** received his M. Sc. and Ph. D. degrees in computer science in 1990 and 2001, respectively. Since 2001 he has been the Assistant Professor at Wrocław University of Technology in Poland. Currently he is working on knowledge integration, multi-agent systems, computational intelligence, adaptive and self-organising systems. Co-editor of 1 book, author of 2 academic books and over 80 other publications including book chapters, journal and conference papers. He is editorial board member of Computational Intelligence magazine and International Journal of Distributed Systems and Technologies. He has served on programme committees for a number of international conferences in the areas of distributed and grid computing, Web technologies, intelligent and multi-agent systems. In 2005 he was rewarded by IBM for the work on Eclipse Didactic Distribution.

**Grzegorz Kukla** is a Ph. D. student in computer science at the Wrocław University of Technology. In 2008, he received his M. Sc. degree and was chosen the best graduate student of Computer Science and Management Faculty of Wrocław University of Technology. He is the author of seven scientific publications on distributed computing and artificial intelligence. His topics of interest are in nature inspired algorithms in distributed systems.