Nature inspired self-healing model for SIP-based services

ZORAN RUSINOVIC
Ericsson Nikola Tesla, Zagreb, Croatia
zoran.rusinovic@ericsson.com

NIKOLA BOGUNOVIC
Faculty of Computing and Electrical Engineering, University of Zagreb, Croatia
nikola.bogunovic@fer.hr

Keywords: autonomic computing, self-healing, SIP, nature inspired computing

The performance characteristics of Session Initiation Protocol (SIP) servers determine user-perceived quality of the services supported by SIP networks. SIP servers therefore must be able to provide service with appropriate reliability.
We present the self-healing SIP model capable of recognizing and restarting failed SIP services without losing active SIP dialogs. Novel approach to an evaluation of the SIP server healthiness has been presented that enables rapid problem detection and consequently quick recovery. Tests show that a proposed model exhibits very promising results with respect to number of successful SIP requests during SIP server operation.

1. Introduction

Session Initiation Protocol (SIP) is a controlling protocol for initiating, managing and terminating IP-based multimedia services across packet networks. In addition to being the controlling protocol between different nodes in IP Multimedia Subsystem (IMS) network, SIP provides building blocks for new media-blending applications and is used for enterprise multimedia applications, multimedia sessions, instant messaging and gaming.

SIP is standardized by the Internet Engineering Task Force (IETF), and has been adopted by 3rd Generation Partnership Project (3GPP) and 3rd Generation Partnership Project 2 (3GPP2) for IMS. An example of the SIP infrastructure is shown in the *Fig. 1.*

We assume the reader to be familiar with SIP, and present here some SIP characteristics only to pinpoint important elements during SIP messages processing.

1.1 SIP characteristics

SIP is an application-layer signaling protocol that can initiate, modify, and terminate interactive multimedia sessions over IP between intelligent terminals. It shares lots of the features that made the HyperText Transfer protocol (HTTP) a success: it is a clear text client/server protocol using Uniform Resource Locators (URL) for addressing. SIP goes beyond the scope of Voice over IP (VoIP) to provide building blocks for new enterprise communication applications:

1. Powerful addressing schemes (URLs) for user-centric services.

2. Features and media negotiation for improved plug-and-play, easily upgraded media-blending applications and terminals.

3. Seamless integration with existing enterprise IP networks and applications: integration with network Domain Name Servers (DNS) and with the corporate directory using the Lightweight Directory Access Protocol (LDAP).

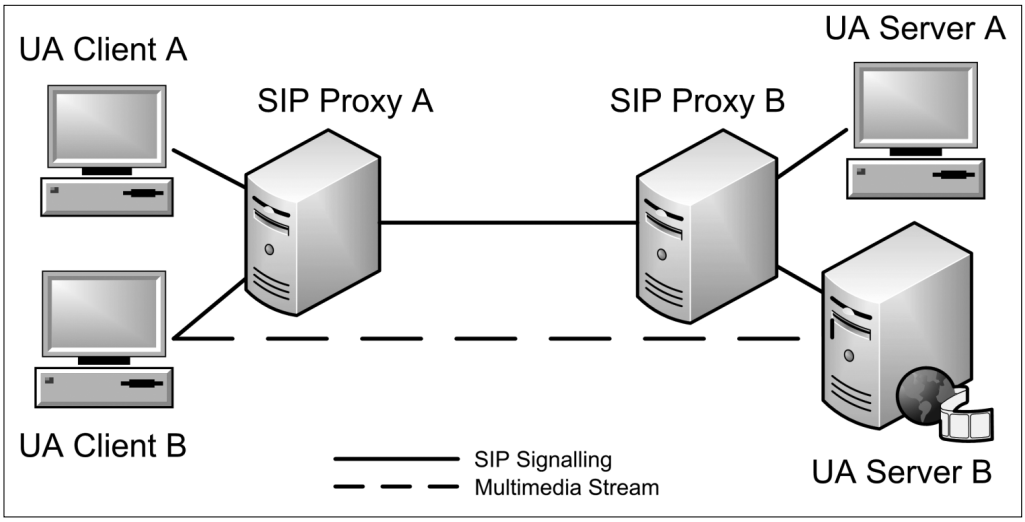


Figure 1. Example of SIP signaling through proxy. User Agent (UA) Server is a SIP service server

4. Built-in extensibility to other information technologies used in enterprises: e-mail, documents transported as Multipurpose Internet Mail Extension (MIME) attachments, etc.

5. Subscription/notification mechanism suitable for transporting user presence and terminal information.

In *Fig. 1* User Agent Client A (UAC A) tries to set up a session with User Agent Server B (UAS B) using SIP. It can be seen that SIP signaling and the multimedia data are completely separated, as SIP protocol is used only to set up or tear down multimedia session. As a result of the separation of SIP signaling traffic and the associated multimedia data stream it might happen that multimedia stream isn't available simply because of the problem with SIP stack and consequently with SIP session establishment, leading to data being unavailable. This will happen even in cases when the data for multimedia services itself is available at the (possibly different) service-providing servers, but because of the SIP signaling problems no connection can be established.

In Fig. 1 this would correspond to the SIP signaling path (marked with full line) between SIP Proxy A and SIP Proxy B being broken and multimedia stream path (marked with broken line) between UA Client A and UA Server B being fine. Although UA Server B itself works fine and despite the fact that no problems exist for providing multimedia stream, as a result of the SIP signaling problems no connection can be established and no service can be provided. This leads to a decreased user-perceived quality or even failure of possibly critical services.

2. Self-healing model for a single network element

When problems occur, traditional approaches for troubleshooting are based on the knowledge and experience of system administrators to discover problems and find ways to correct them. Unfortunately that approach, in addition to being laborious, is time-consuming and can lead to the SIP service being unavailable for a long period of time.

Self-healing is the property of any device or system to recognize that it is not operating correctly and to make any necessary adjustment needed to restore itself to normal operation. In a previous paper [2] the authors discussed the SIP service self-recovery model based on the monitoring of SIP messages exchange between SIP agents that was focused on the ability to efficiently utilize self-healing environment for SIP-based services within a single SIP network element. This article extends that work in such a way that it addresses cross-server healing between multiple network elements in the SIP-based networks. Together with the self-protecting SIP stack capability described in [4] we believe this to be a step towards an autonomic environment for SIP-based services.

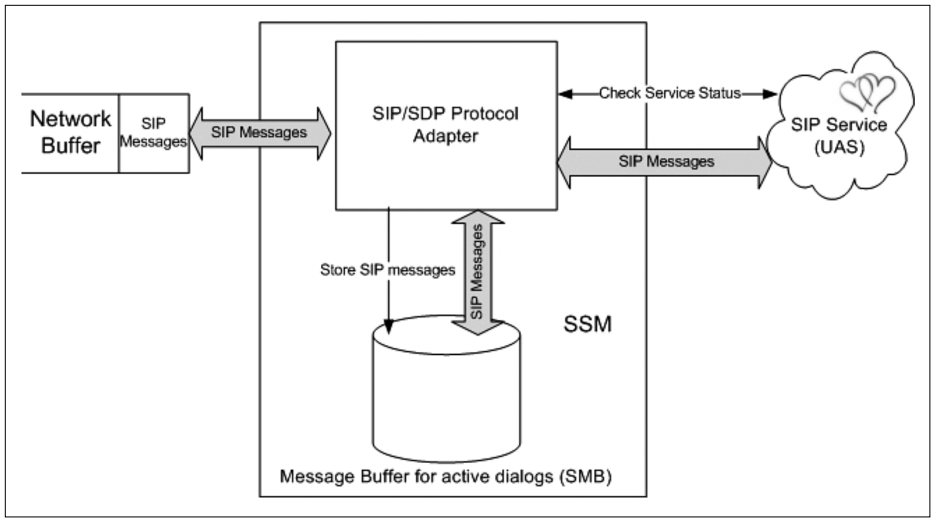
Our approach is based on the heartbeat monitoring with a purpose to detect whether monitored services are working or not. Heartbeat monitoring is an approach that can be seen as type of environment awareness since it provides awareness about the health status of system parts [5]. When the SIP Service fails, its peer SIP User Agent (i.e. peer SIP node) will detect this by expiration of timers defined as the part of SIP protocol, however for the failed service to continue its operation local monitor is needed that can detect that the SIP service has failed and that will try to restart it. The following sections will describe self-healing model based on the Windows Server 2008 operating system, but most conclusions therein can be applied to other operating systems as well.

2.1 SIP Service Monitor (SSM)

Fig. 2 depicts high-level relationships among the framework's main components. SIP services Monitor (SSM) is the component that implements monitoring and recovering for SIP services. If the service is recognized as failed it can be terminated and restarted. In the Windows environment the common approach to detect if the application is blocked is to send WM_NULL message to the suspected application by using the API call SendMessageTimeout().

The WM_NULL message performs no operations, and the recipient will ignore this message, however if the target application is non responsive the API call TerminateProcess() can be used to kill the hung instance of the application. This method is used, for example by the Windows Task Manager to recognize not responding applications and is also used as a base for some auto-

Figure 2.
Relationships among the main components of the SIP self-healing model



nomic self-healing tools frameworks [6]. There are, however, few problems with this approach that makes it inapplicable for SIP self-healing framework (as well as for many other cases).

In Windows environment there are two kinds of threads:

1. User interface threads that create its window (which can be invisible) and have GetMessage() loop which is used to respond to user actions.
2. Worker threads which do not create its own window and are used to do a compute intensive job in the background.

Windows applications usually have a single thread used for all user interface components which creates one or more worker threads. User interface thread typically runs with higher priority than worker threads, so that user interface is responsive to the user while the working thread is doing a background job. Typical example for this are SIP applications which usually create a worker thread for parsing SIP messages and running SIP protocol state machines in the background during the times when there is no user input. Being a preemptive OS, upon reception of the WM_NULL message Windows will suspend the lower-priority worker thread (if still running) and assign the CPU to the higher-priority user interface thread that responds to SendMessage Timeout(). For those reasons if it happens that SIP message parsing or state machine handling fails, using the above described approach, SIP application will appear healthy despite the fact that there is a problem with SIP protocol handling thread, and that the only healthy part of an application is an user interface.

Another problem lies in the fact that this method is not applicable to applications running as Windows services which by default don't have message loops and run in different desktop. This is because services run in their own security context and not the context of the user that is logged in. Therefore services also run in their own virtual desktop which means that they cannot display any blocking user interface nor can they interact with the user (since normally hooks are only global within a single desktop).

In this work we propose another approach based on monitoring applications ability to process SIP protocol messages. In addition to avoiding all the problems related to the first method, this approach can monitor changes in processing times of messages and reflect more accurately applications behavior and its health.

2.2 SIP flow monitoring in SSM

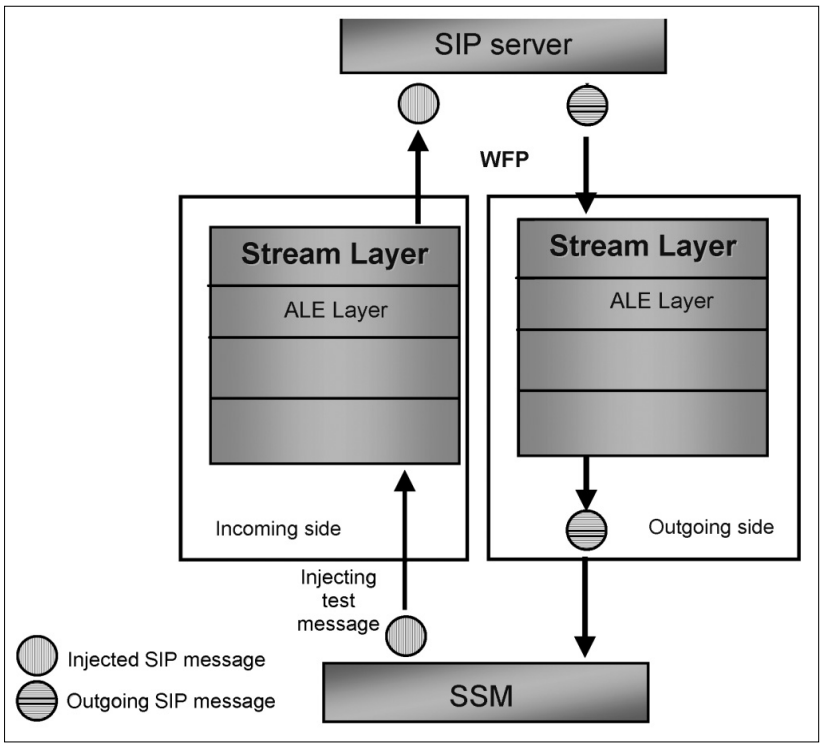
With the introduction of the new networking stack in Windows Vista and Windows Server 2008 with a single transport and framing layer, a new API known as a Windows filtering Platform (WFP) has been introduced. WFP allows, among others, to filter and modify packets, monitor and authorize connections at different layers in TCP/IP processing path.

By providing a simpler development platform, WFP is designed to replace previous packet filtering technologies such as Transport Driver Interface (TDI) filters, Network Driver Interface Specification (NDIS) filters, and Winsock Layered Service Providers (LSP).

As seen in *Fig. 3* on the incoming side all packets arriving at SIP port are monitored on the Stream Layer in the packet processing path using management API functions (FWPM) for the managing of the filter engine (FWPM_LAYER_STREAM_V4 and FWPM_LAYER_INBOUND_TRANSPORT_V4 for TCP and UDP respectively). Upon the reception of the SIP message a WFP callout function is called which determines if the received message is a request or a response (callout functions provide functionalities that extend the capabilities of the WFP, and can be registered at any layer). If a message does not begin with "SIP/" prefix, then the message is assumed to be a request which (even if incorrect one) requires a final response. SSM can operate in two modes, each providing different levels of recovery. In the basic mode WFP is used to monitor outgoing SIP traffic. When there is no outgoing traffic for a predefined minimum time period, SSM will use WFP to inject a testing SIP INVITE message on receive path, consisting only of mandatory headers excluding Call-Id header. If SIP UA server is responsive it should reply with "400 Bad Request" response, which will be caught by SSM at the SIP UA server's outgoing queue.

To monitor health of the SIP UA server more thoroughly, processing times of the injected messages can

Figure 3.
SSM operates on the WFP Stream layer



be stored and compared (since processing time should be in a microsecond range this can be done using timestamp counter in a CPU). If despite this testing SIP INVITE message being injected no outgoing SIP message is generated from the SIP UA server, SSM will conclude that server has failed and will try to restart it. Otherwise SIP UA server is assumed to work correctly, and the generated response is simply dropped in SSM.

In the advance mode in addition to the basic operation, each incoming SIP messages are cloned into the SSM Message Buffer (SMB) together with the hash value of the corresponding dialog as seen in *Fig. 4.* During the SIP UA server restart SSM is used to listen on the server's socket and to store all incoming requests into the SMB. Dialog hash value (to be used for comparing dialogs by numbers) is calculated using the 64-bit FNV1 algorithm using Call-Id value, remote tag and local tag of the SIP request as an input. Since initial INVITE messages do not have remote tag (which is provided by the SIP UA server itself in the response) the hash value of such request is calculated after the response has been generated. SIP requests are held in the SMB for the whole duration of the dialog, and are deleted from there only after the final response on BYE request has been sent by the SIP UA server. If sometime during the operation SSM detects problems with SIP UA server, in addition to restarting the server, SIP messages from SMB are used to re-initialize the dialog and transaction states of the server into states server had before restart. During this re-initialization phase initial INVITE requests from SMB are injected into the incoming buffer. Callout function is used to detect the new local tag value from the first non-100 provisional response generated by the SIP UA server. If it differs from the old one (from the pre-restart phase), all other SIP requests of the same dialog are injected into the incoming buffer with this new local tag. In addition to this, all responses to injected requests are dropped by SSM, to prevent SIP UAC receiving responses to old requests.

Reason behind this is that remote SIP UAC will have dialog state machines driven by old tag values and the local SIP UA server has them driven by new local tag values. Therefore it is necessary to make an appropriate mapping for all new SIP requests.

During the advance mode of operation prior to buffered INVITE requests being injected into the incoming buffer, each SIP request from SMB is internally checked in the SSM. This is done in the SSM's SIP message recognizer, whose purpose is to prevent sending incorrect SIP message to SIP server. Recognizer is a piece of code that scans and parses the input and recognizes whether it is in the language of the grammar, but does not produce an abstract syntax tree or any other form of output that represents the contents of the input. Because of this intertwining of lexical analysis and parsing of SIP messages, instead of having separate scanner, integrated recognizer has been used. This approach has a number of advantages including discarding of the lexical disambiguation by means of the context in which a lexical token occurs. Consequently the possibly complex interface between scanner and parser is removed, and both lexical and syntax checking are integrated into a single analysis phase. This approach is sometimes called scannerless parsing; however this term is somewhat misleading since all of the characters are anyhow scanned from the input buffer. Implementation difference is that instead of having one scanning function that accumulates characters into tokens, there are multiple functions that can read characters and then try to match it against some grammar construct.

If the SIP message is recognized as a correct SIP message it is then injected into the incoming buffer, otherwise it is dropped in the SSM itself and the "400 Bad Request" response is sent to the peer UA directly from the SSM. This way we prevent SIP server from going into cyclic restarts if the particular SIP request is causing server restart.

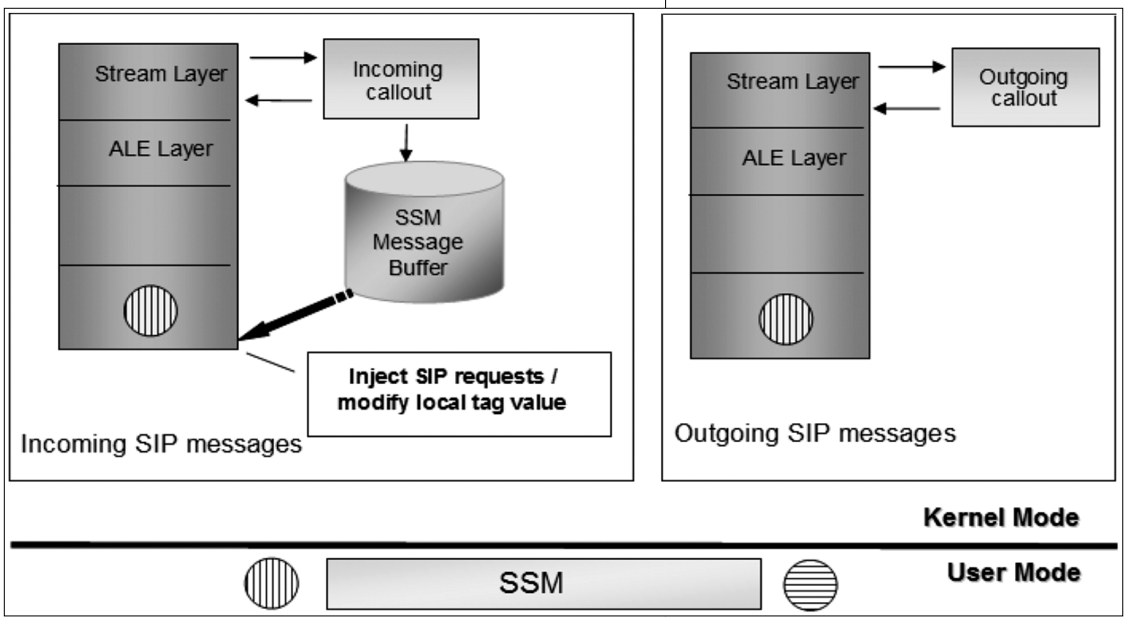


Figure 4. SSM operating in the advanced mode. Each incoming SIP request is cloned into the SSM Message Buffer.

The cost of this virtually fault free operation is increased processing time at both SIP message receiving and sending sides. Calculating dialog hash value requires not only message cloning but also finding required SIP message elements during message receiving and sending. This is done using WFP callout functions in Windows kernel mode.

3. Self-healing in the SIP-based networks

In the previous section we have described a model of self-healing within a single SIP-based network element. In this section we will extend that model to the SIP-based network architecture.

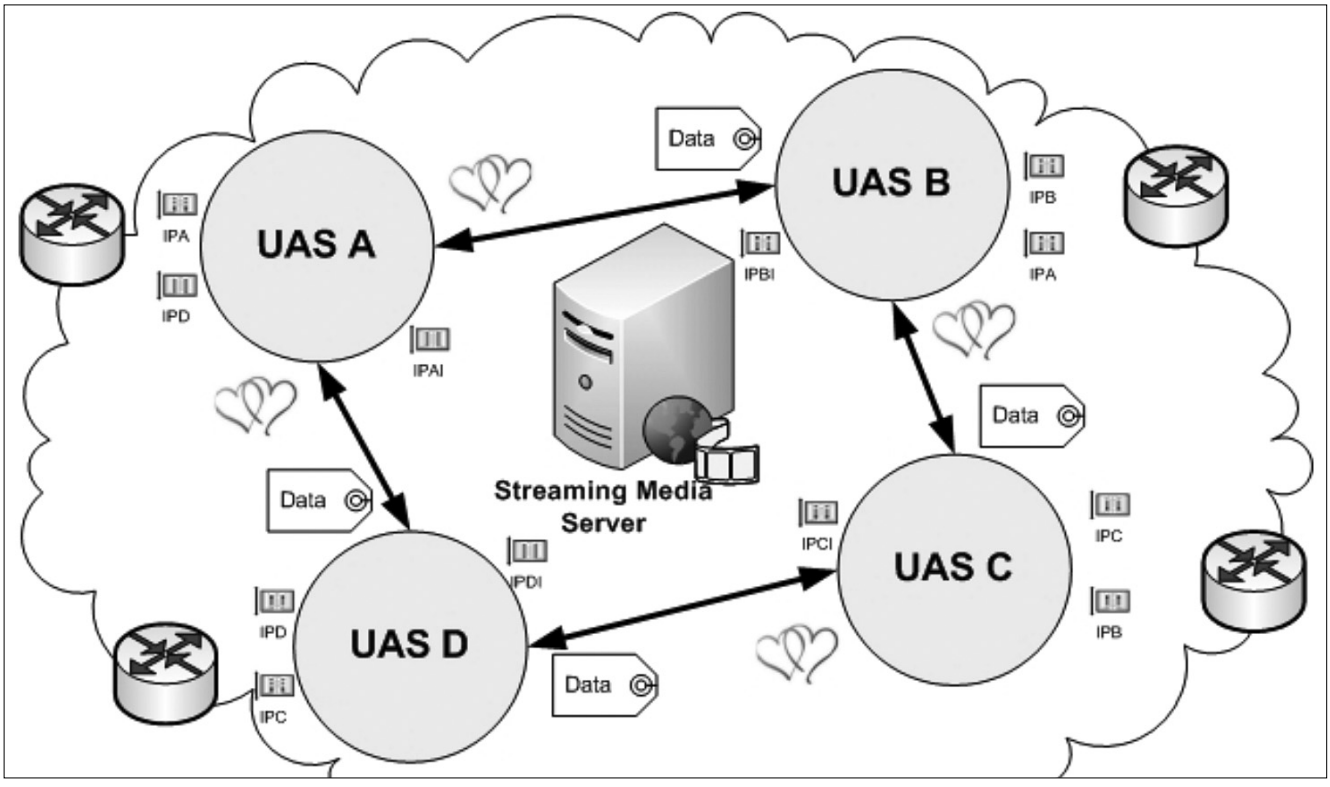
To solve the problem of service availability the current practice (as implemented by some providers) is to dedicate another SIP node as a backup of a primary SIP node in such a way that either each primary SIP node has its backup node (2*N model) or in a way that a set of SIP nodes is dedicated as backup set to relay the primary ones out of service. The problem with both of these approaches is that in addition to requiring large number of nodes to being deployed, the service becomes unavailable during the transition phase needed for the backup server to become operational. Moreover, neither of these approaches is capable of preserving SIP sessions that were in a set-up phase at the moment when a failure has happened [1].

We have based our approach loosely on the unilateral mode of anycast-based model for service continuity in IMS networks [1]. In this mode public interface of the UAS consists of two IP addresses, a primary and a secondary IP address. This secondary address is the primary address of another UAS. Problem with that model is that in case of node failure it does not preserve SIP sessions in progress. Consequently the service will be disrupted for all sessions in progress when a given UAS encounters failure. With our approach UAC perceives undisrupted level of service despite the UAS failure.

To ensure this, similar principle as described in the single network element scenario has been used. Unlike with single network element where the heartbeat monitor is implemented in the element itself, in this configuration the heartbeat monitor is implemented in the partner UAS, that is called guardian UAS (conversely the UAS being monitored is called guarded UAS). The configuration of such self-healing network is shown in the *Fig. 5.*

The network model in Fig. 5 is configured as an NK network with K=1, which means that every UAS sends heartbeats to only one partner UAS, namely the guardian one (in the Fig. 5. UAS B monitors heartbeats of UAS A, UAS C of UAS B, and so on). In addition to two public addresses each UAS has a dedicated inner interface (IPxI) that is used to convey parsed content of SIP messages, in the internal format, to its guardian node. Each UAS announces its primary and secondary address to

Figure 5. Configuration of the self-healing network model corresponds to a NK network (K=1). Heartbeat monitor is implemented in the guardian UAS. Each UAS has its own guardian UAS (in this case UAS B is a guardian UAS of UAS A, UAS C is a guardian UAS of UAS B and so on). Each UAS announces its address to router over OSFP.



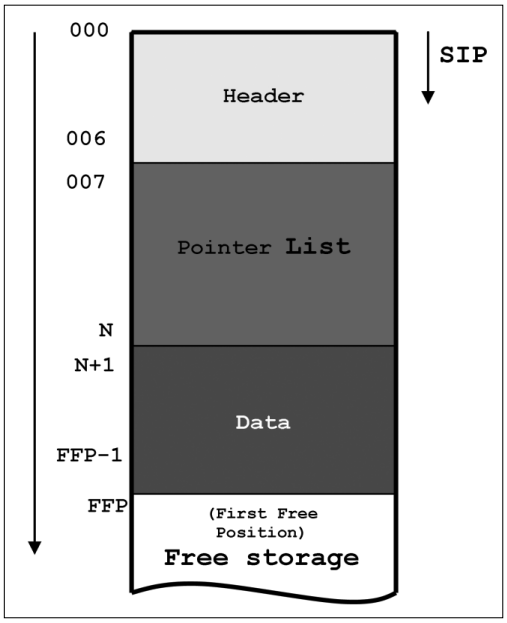


Figure 6.
Internal format used to convey parsed SIP messages between the guardian and guarded UAS. It consists of the Header part that conveys dialog and transaction hashes and Pointer lists that points to headers which are stored in "Data" part.

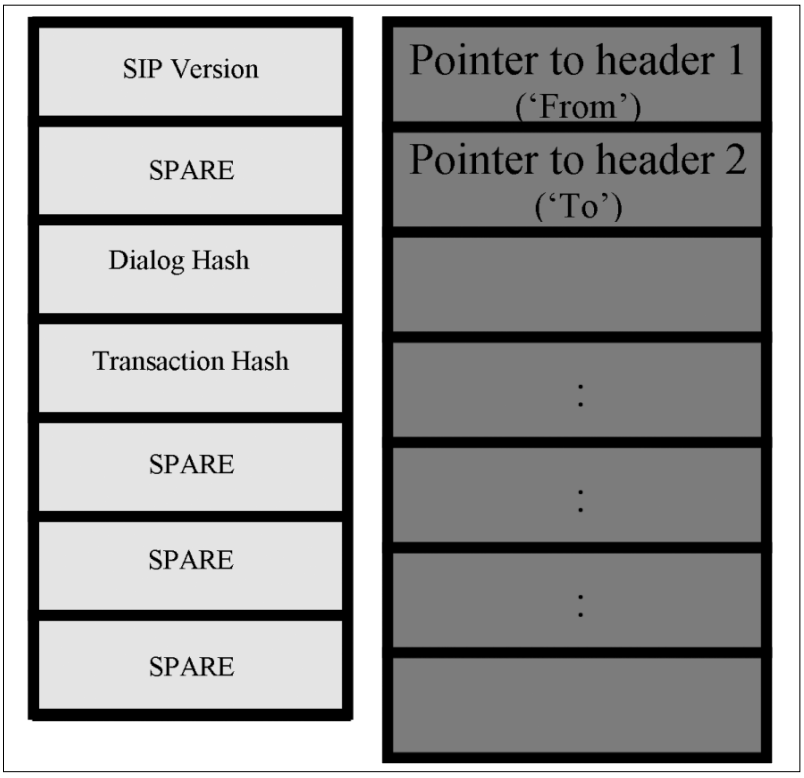


Figure 7. Header part of internal format (left) and Pointer list (right)

the router using Open Shortest Path First (OSPF) protocol (or another IGP protocol), so if the server dies, the router will remove it from an announcement.

3.1 Heartbeat monitoring in network

Heartbeat monitoring works as follows: when the UAS receives message from the UAC the message is parsed and converted into the internal format shown in the *Fig. 6* and *Fig. 7.* Internal format consists of the fixed header which conveys hashes of the particular dialog and transaction and a fixed number of pointers that point to the particular SIP message headers which are stored in the data part of the internal format. As a consequence of using an internal format a guardian node does not need to reanalyze the whole message but can directly use parsed data from the internal format in case that it must take over the function of its guarded node (this mechanism is similar to the one in MPLS where edge devices use labels instead of IP addresses to further forward them). The message is then sent to its guardian node in its internal format using nodes inner interface.

If the SIP method received in message is BYE or CANCEL guardian node will use dialog hash value to find and delete this session from the list of active sessions, otherwise it will simply store the received message in the SMB. Those messages will then be used to re-initialize dialog at UAS (in case of the failure of the guarded node) in the same ways as it is described for the single network element.

Previously described mechanism for heartbeat pacing, when no data is received from the guarded UAS for a certain minimum time period, is applied here as well. Guardian UAS will use SSM to send a testing SIP INVITE message to the guarded UAS, consisting only of mandatory headers, excluding Call-Id. If alive guarded UAS will response with "400 Bad Request" that will be sent to back to the guardian UAS and ignored. When the guarded UAS (e.g. UAS A in Fig. 5.) gets out of service secondary addresses of the guardian UAS (e.g. UAS B) are announced in IGP (Interior Gateway Protocol) and, as already described for a single network elements, SIP messages from SMB (of the guardian UAS) are used to re-initialize the dialog and transaction states of the UAS for active dialogs.

In this configuration UAC that were attached to the failed UAS will be dispatched to the UAB in a transparent way. For the period during which the failed UAS is out of service a guardian UAS might temporarily encounter a burst of SIP messages. To compensate that we propose the usage of simple cellular automata (CA) based model.

3.2 CA model of networks dynamic

We assume the reader to be familiar with Cellular Automata, and present here only some basic elements.

Cellular automata, firstly introduced by Ulam and Von Neumann [8], are a special class of finite automata that can be described by the 3-tuple of *Eq. (1).* They contain large numbers of simple identical components with only local interconnections.

$$A = (S, N, \delta) \tag{1}$$

In the above equation S is a nonempty set, called the state set, $N \subseteq \mathbf{Z}^2$ is the neighborhood, and $\delta : S^N \rightarrow S$ is the local transition rule.

A lattice of N identical finite-state machines (i.e. cells), each with an identical pattern of local connections to other cells for input and output, is called a cellular space. Each cell is denoted by an index i and its state at time t is denoted s_i^t (where $s_i^t \in S$). Cell i together with the cells to which cell i is connected is called the neighborhood η_i^t of the cell i. Local transition rule $\delta: S^N \rightarrow S$ gives the update state s_i^{t+1} for each cell i as a function of η_i^t. Typically CA works in a discrete manner. That is to say time goes step by step and a global clock provides an update signal for all cells.

The proposed models consists of a one-dimensional automata with three cells per each UAS and is similar to the Hodgepodge Machine of Gerhardt and Schuster which was used to simulate oscillating chemical reactions [9].

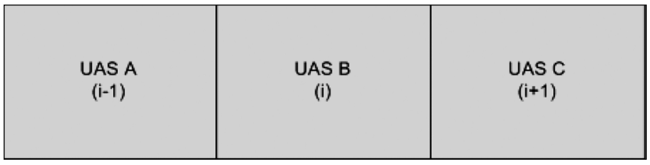


Figure 8.
One-dimensional neighborhood of the CA consisting of the UAS itself (UAS B in the Fig. 5) and its guardian server (UAS C in the Fig. 5) and guarded server (UAS A in Fig. 5)

Neighborhood of the CA consist of the UAS itself, and its guardian and guarded UAS, as displayed in the *Fig. 8*. The updating of cell sites is done asynchronously during the heartbeat monitoring at each heartbeat. Each cell can be in one of the five states:

- 0 = healthy
- 1 = infected
- 2 = ill with high load
- 3 = ill with overload
- 4 = death

Cell (UAS) is healthy if it receives traffic only on its primary IP address. If the secondary IP address is active as well we consider such UAS infected since it has to deal with excessive traffic of its guarded UAS. However in this state UAS handles this additional traffic in such a way that a regular traffic has not escalated into the high-load or overload traffic. In states 2 and 3 UAS has to handle excessive traffic but in such a way that this additional traffic is causing a high-load or overload, respectively (difference being that in high-load situation UAS receives more messages that it can process within a given period of time but it retains control of how to handle them, while in the overload situation messages are lost without control). Cell is dead if the corresponding UAS does not return heartbeats.

Formally we define this CA as follows:

$$\begin{gathered} A = (S, N, \delta) \text{ with } S = \{0,1,2,3,4\} \text{ and} \\ N = \{c_{i-1}, c_i, c_{i+1}\} \\ c_{i-1} = \text{guardian UAS} \\ c_i = \text{observed UAS} \\ c_{i+1} = \text{guarded UAS of the } c_i \end{gathered} \tag{2}$$

Transition rule is defined as follows:

$$\delta(c_i^t) = \begin{cases} 0 & if\ c_{i-1}^t = 0\ and\ c_{i+1}^t = 0 \\ 1 & if\ c_{i-1}^t \geq 1\ and\ c_{i+1}^t = 0 \\ 1 & if\ c_{i-1}^t = 4\ and\ c_{i+1}^t \neq 4 \\ c_i^t & otherwise \end{cases} \tag{3}$$

If all the UAS from the CA neighborhood at time t are healthy (meaning that they are in state 0) then at time t+1 they will remain in state 0 and no action is taken. However if at time t a guarded UAS is not healthy and provided that a guardian UAS itself is healthy then at time t+1 a guardian UAS will change its state into the state 1 and will take over part of the traffic of its guarded UAS (this transition takes place regardless of the state of a guardian UAS when the guarded UAS is dead and a guardian UAS is not). In all other cases the state of the guardian UAS remains the same.

This transition rule ensures that for situations where failing of the guarded UAS (e.g. UAS A in Fig. 5) causes excessive traffic at the guardian UAS (e.g. UAS B in Fig. 5) its guardian UAS (UAS C in Fig. 5) jumps in by announcing in IGP its secondary IP address and provisioning that some UAC traffic is dispatched to it. Naturally, if the excessive traffic persists too long because of (multiple) nodes failure, the risk of overload remains.

4. Testing and results

To test the efficiency of the described self-healing solution as well as the cost of SIP message preprocessing we have tested solution in our local lab environment. Test was run on an isolated Ethernet network using dual-core AMD Opteron processor running at 2.4 GHz with 8 GB of RAM for running SIP Service and SSM. From the SIP traffic generator SIP requests are sent according to a standard proxy 200 scenario. For testing self-healing within the network configuration as depicted in Fig. 5 has been set up.

Request intensity was 10, 50, 70 and 100 requests per second respectively, with each of these four different SIP loads running for half an hour. SIP UA server was modified to fail (enter an infinite loop) every 10 minutes. *Fig. 9* shows number of successfully handled requests without using SSM, using SSM in basic mode, and using SSM in advance mode. For the network self-healing testing a random UAS would fail every 10 minutes.

As seen in Fig. 9 using SSM even in the basic mode increases percentage of successful SIP requests to ~99.998 percent. The difference to 100% is lost on requests being processed or just sent from SIP UAC at the time of SIP UA server restart. In the advance mode, all the SIP requests being processed or sent from the SIP UAC were available locally in the SMB and were used to reinitialize all UA server state machines to the pre-restart baseline bringing successfulness to 100 percent.

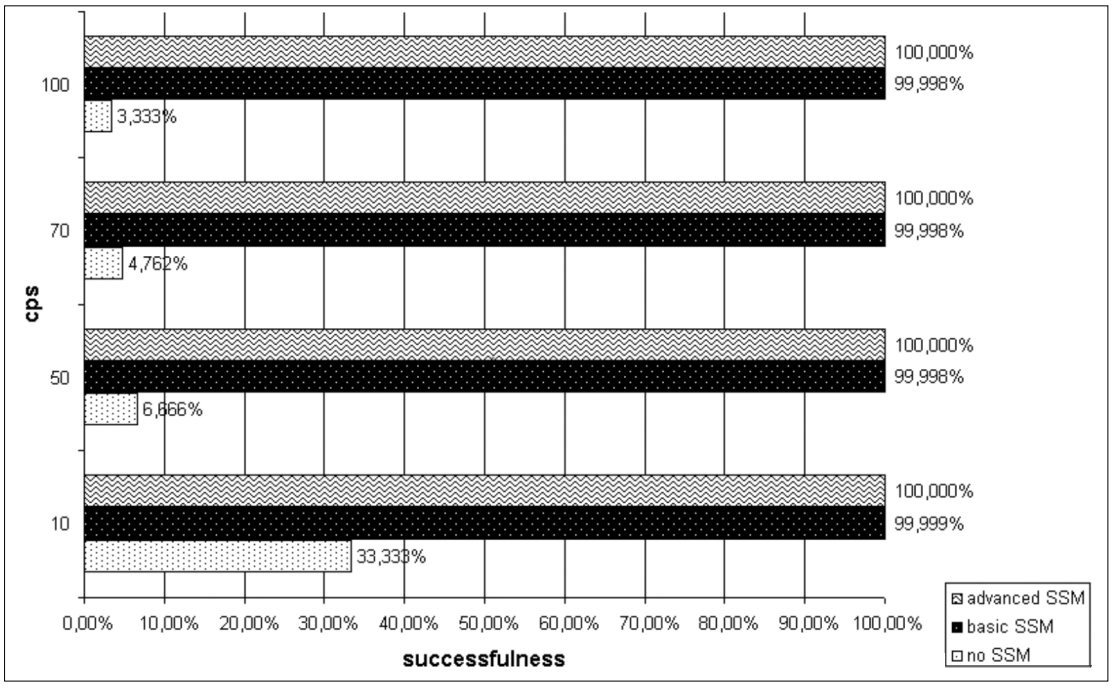


Figure 9. Results showing number of successfully processed SIP request without SSM, with SSM working in basic mode and with SSM working in advanced mode

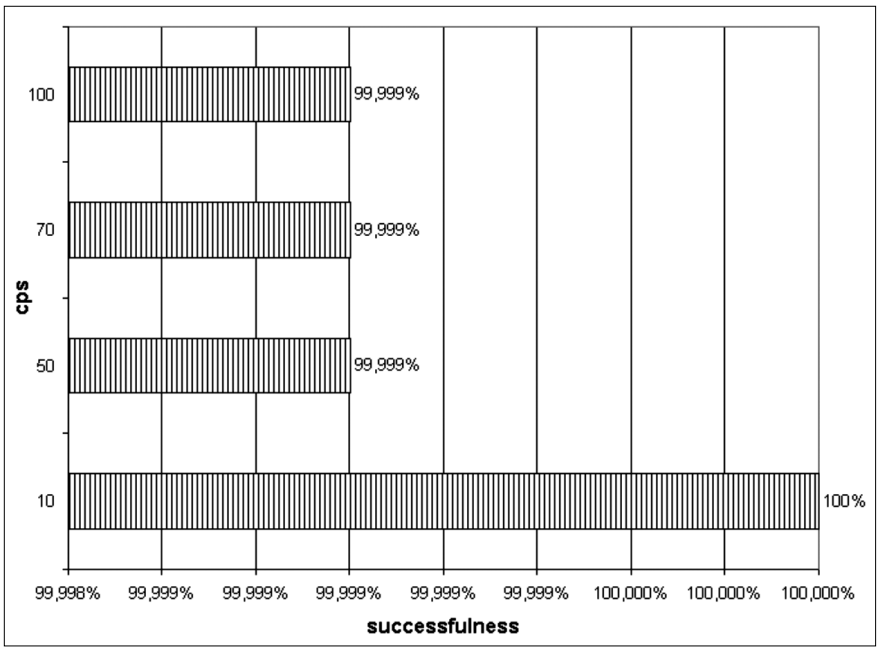
Results for the self-healing recovery in the network are shown in *Fig. 10.* In this case during high-load traffic this approach reaches five nines of successfulness. We attribute this to small portion of SIP messages being inevitably lost during route reconfiguration after UAS has failed.

The cost of the SSM operation is shown in the *Table 1.* Comparing the time needed to process a SIP message without SSM active and with SSM active it can be seen that in the basic mode which provides health-monitoring and service recovery (but no recovery of active dialogs and requests that were in progress during restarts) SSM adds ~10% of overhead to the processing times.

Message	Bytes	SDP	No SSM	Basic SSM	Advance SSM
INVITE	931	yes	8.4 μs	9,1 μs	11,8 μs
100 Trying	320	no	2.9 μs	3,1 μs	4,4 μs
183 Progress	854	yes	7.8 μs	8,4 μs	15,1 μs
PRACK	386	no	3.5 μs	3,9 μs	5,4 μs
200 OK	330	no	3.0 μs	3,4 μs	4,3 μs
UPDATE	731	yes	6.6 μs	6.9 μs	10.1 μs
200 OK	721	yes	6.6 μs	6.8 μs	10,2 μs
183 Progress	622	no	5.4 μs	6,1 μs	8,2 μs

Table 1. Messages decoding times

Figure 10. Results showing number of successfully processed SIP request in the network using the SSM



However in the advance mode which provides recovery of all dialogs the overhead is ~50% because of the additional time needed to copy each request into the SMB and to calculate corresponding hash value. Despite the somewhat increased processing times, numbers from Table 1 demonstrate the capability of our technique to handle SIP processing requirements of non-trivial size for real SIP-based systems. It is worth to mention that the demonstrated efficiency could further be improved by optimization that should be applied to memory handling routines.

Similar results are obtained for network recovery, shown in *Table 2,* which demonstrates that the app-

Message	Bytes	SDP	Processing time w/o heartbeat monitoring	Processing time with heartbeat monitoring
INVITE	931	yes	8.2 μs	12,1 μs
100 Trying	320	no	2.9 μs	4,6 μs
183 Progress	854	yes	7.8 μs	15,8 μs
PRACK	386	no	3.7 μs	15,7 μs
200 OK	330	no	3.2 μs	4,9 μs
UPDATE	731	yes	6.8 μs	10.9 μs
200 OK	721	yes	6.5 μs	10,9 μs
183 Progress	622	no	5.5 μs	9,1 μs

Table 2. Messages decoding times for network

roach with internal format distribution between two UAS is very efficient and introduces a very slight overhead.

Finally, the effectiveness of the CA model dynamics to compensate the traffic outbursts is shown in *Fig. 11.* UAS were dimensioned to handle 120 calls per second (CPS) in high-load traffic without messages being dropped. As seen in the picture after the guarded UAS fails without CA assistance almost 30% of messages were lost during 70 CPS traffic and almost 90% during 100 CPS traffic. However with the CA assistance the number of dropped messages decreases to ~3% and ~9% for 70 and 100 CPS respectively. Such results are understandable since with CA assistance excessive message traffic will be dispatched between several UAS in a transparent way.

5. Summary and conclusion

In this work we have presented an approach to self-healing SIP networks. New measure for evaluating SIP nodes health, based on a SIP requests processing capabilities, has been proposed. By experimental measurements it is shown that the proposed solution is very efficient in self-healing for both single network element, as well as in the SIP-based network and the obtained results are very promising. With the proposed approach service providers can ensure that outputs are not fuzzy and is always within service level agreements Further self-healing capabilities could evolve under this model including media server recovery to provide self-healing not only for signaling traffic, but for media traffic as well.

Authors

ZORAN RUSINOVIC received M.S. in Computer Science from the Faculty of Computing and Electrical Engineering of Zagreb University in 2004 and is currently a Ph.D. candidate at the Computer Science department. In 2004 he joined Ericsson Nikola Tesla company in Zagreb, Croatia, where he is currently a senior consultant for enterprise IT solutions. He has been engaged in many R&D projects in Croatia, Sweden and Germany, working in the area of network system engineering, network evolution, QoS/ TE managements and service architecture. For the past few years his focus has been IMS architecture and end-to-end QoS for multimedia delivery in SIP-based networks. His research interests include next-generation networking, biologically inspired computing, self-star and autonomic systems.

NIKOLA BOGUNOVIC graduated in 1967 from the Faculty of Electrical Engineering, University of Zagreb where he was awarded M.Sc. and Ph.D. degrees in 1971 and 1984, respectively. From 1968 until 1971 he was a research assistant at Institute Rudjer Boskovic, Zagreb. In late 1971, he was assigned a position of visiting research associate to UKAEA, Cullham Lab., England. Upon his return to Rudjer Boskovic in 1972 he was a principal investigator in various projects. At Vanderbilt University, USA. he was engaged as full time visiting associate professor in 1985. In 1990 he was assigned a position of co-associate professor to Faculty of Electrical Engineering and Computing, University of Zagreb. In 1996 he was appointed head of a Division of Electronics at Rudjer Boskovic, and occupied a position of scientific advisor in 1998. In 1999 he was assigned a position of full professor at Faculty of Electrical Engineering and Computing where he headed the Department of Intelligent Systems from 2005 to 2008. Professor Bogunovic is a full member of the Croatian Academy of Technical Sciences and has published over 100 scientific and professional papers. His scientific interests include computer based instrumentation systems, intelligent systems and methodologies for complex computer system design.

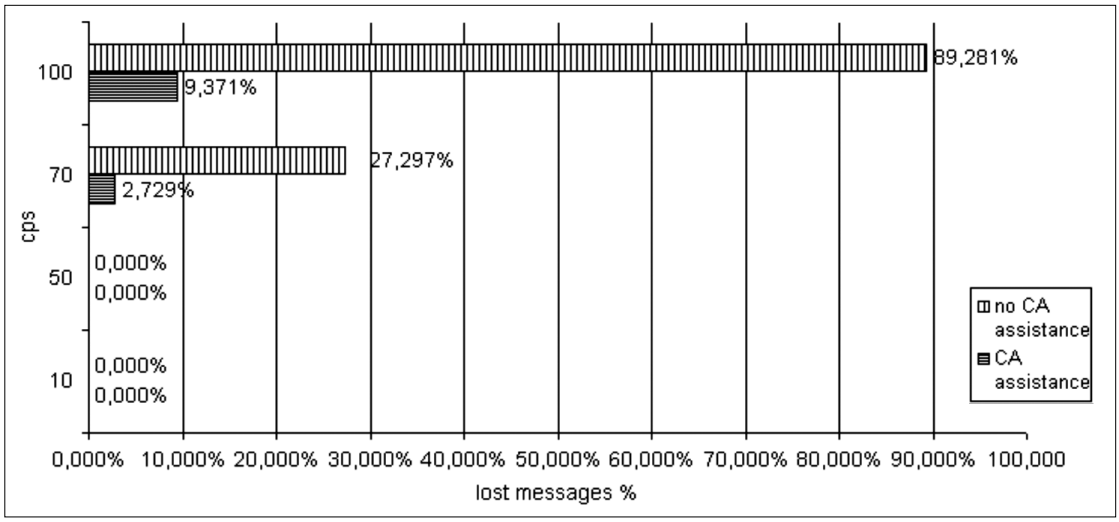


Figure 11. Percentage of lost messages with and without CA assistance during guarded node failure. UAS were dimensioned to be able handle 120 CPS in high-load traffic.

References

[1] Boucadair M.,
"Introducing Autonomous Behaviors into IMS-Based Architectures",
In: Autonomic Computing and Networking, M.K. Denko, L. Tianruo Yang, Yan Zhang (Eds.), Springer-Verlag, Berlin Heidelberg, pp.155–178, 2009.

[2] Rusinovic, Z., Bogunovic N.,
"Self-healing Model for SIP-Based Services",
In Proc. of the 10th Int. Conf. on Telecommunications (ConTEL 2009), pp.375–379, 2009.

[3] Rosenberg, J., Schulzrine, H., Camarillo, G.,
"SIP: Session Initiation Protocol", RFC 3261.

[4] Rusinovic, Z., Bogunovic N,
"Self-Protecting Session Initiation Protocol",
In: Lecture Notes in Artificial Intelligence, Vol. 5177, Proc. of the 12th International Conf. Knowledge-Based and Intelligent Information and Engineering Systems, I. Lovrek, R.J. Howlett, L.C. Jain, (Eds.), Springer-Verlag, Berlin Heidelberg, pp.717–724, 2008.

[5] Sterritt R., Bustard D.W.,
"Towards an Autonomic Computing Environment",
In Proc. of the 14th Int. Conf. on Database and Expert Systems Applications (DEXA), pp.699–703, 2003.

[6] Sterritt R., Chung S.,
"Personal Autonomic Computing Self-Healing Tool",
In Proc. of the 11th IEEE International Conference on the Engineering of Computer-Based Systems ECBS, pp.519–527, 2005.

[7] Kuthan, J.,
"Accelerating SIP,"
SIP 2002, Paris, France, 2002.
http://www.iptel.org/

[8] J.V. Neumann,
"The Theory of Self-Reproducing Automata",
A.W. Burks (Ed.), Univ. of Illinois Press, Urbana and London, 1966.

[9] Gerhardt, M., and Schuster, H.,
"A cellular automaton describing the formation of spatially ordered structures in chemical systems",
In: Physica D, Vol. 36, pp.209–221, 1989.