

# A Bio-Inspired Approach for Self-Protecting an Organic Middleware with Artificial Antibodies

Andreas Pietzowski, Benjamin Satzger, Wolfgang Trumler, Theo Ungerer

Institute of Computer Science – University of Augsburg  
86159 Augsburg, Germany  
{pietzowski, satzger, trumler, ungerer}@informatik.uni-augsburg.de

**Abstract.** Our human body is well protected by antibodies from our biological immune system. This protection system matured over millions of years and has proven its functionality. In our research we are going to transfer some techniques of a biological immune system to a computer based environment. Our goal is to design a self-protecting middleware which is not vulnerable to malicious events. First off this paper proposes an artificial immune system and evaluates optimal parameter settings. This shows the correlation between the size of a system and the length of the receptors used within antibodies for an efficient detection. Our tests showed that the recognition rate of unknown malicious objects can reach up to 99%. Further on we describe the integration of the immune system into our organic middleware  $OC\mu$  and afterwards we propose techniques to minimize the memory space needed for storing the antibodies and to speedup the time needed for detecting malicious messages. We obtained a space minimization by 30% and gained a speedup of 30 with execution time optimization.

## 1 Introduction

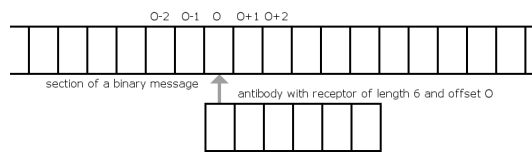
To secure computer systems, current protection applications (e.g. virus scanners) have to be aware of signatures from viruses or worms that occurred previously. Due to that restriction they cannot recognize or handle brand new intruders that are not already stored in a database. Computer immunology opens up new ways and methods to recognize new intrusions like our biological immune system does [2] and fits well into the research fields of autonomic [12] and organic computing [13] that explore self-organization techniques to achieve computing systems that are self-configuring, self-optimizing, self-healing, and self-protecting. In our organic ubiquitous middleware research [16] we investigate self-protection techniques to cope with intentionally or unintentionally malicious peers or services. The biologically inspired technique of computer immunology extracts ideas from our human immune system to develop an artificial counterpart [4]. Thus the following approach is a first protection step which can detect intrusions within a system with high message activity in a fast way. The main goal is to develop a whole self-protecting system like our biological immune system which is permissive to good-natured middleware services and messages but can detect appearing

malicious events. This immune system is implemented in our organic middleware to evaluate its self-protection behavior in a real environment.

The next section confronts the biological with the artificial immune system and shows the different preferences and their effectiveness. We continue with optimizing the system as well for memory requirements as also for execution time minimization. Subsequently we discuss the implementation of the self protecting system and its components in the organic middleware. In the end we mention related researches and end up with conclusions and future work.

## 2 Transition from Biological to Artificial Immune System

Lets first have a look at the functionality of the biological immune system which is present in every animal and human being. The basic requirement of such an immune system is to distinguish between harmless objects called *selfs* and harmful objects or antigens called *non-selfs* [9]. Matching works due to different protein patterns on the surface of the unknown objects and the antibodies. In our body we have an instance named *thymus* which is aware of all selfs known in the body. Antibodies are continuously created with random protein surfaces and get singled out with *negative selection*. This means if such a cell is activated in the thymus it will be destroyed [6] otherwise it will be released to the body to protect it against a specific type of intruder. This biological immune system is extremely distributed besides the centralized work of the thymus [15].



**Fig. 1.** Example how an antibody docks to a message and starts comparison

In a computer-based immune system the working domain concerns bit strings instead of proteins. The goal in an artificial immune system is also to distinguish between foreign non-self messages and messages which are tolerated in the system because they belong to self. It also should be distributed that the middleware does not end up unprotected if one central detection node in the system gets disabled or unreachable. To avoid this we abided by the highly distributed biological paradigm. Thus we developed antibodies which are represented by a short bit string of length  $R$  which embodies the receptor. Additionally the antibodies have a specific offset where they start their comparison to the messages (see figure 1). Instead of waiting for all newly created antibodies until they perambulated the thymus we preferred to generate them in a structured way. This works because the middleware is aware of all its self messages and thus it can create all receptor patterns not used by any self message at a specific offset.

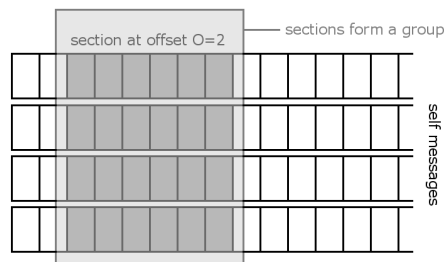
### 3 The Design of the Artificial Immune System

With the approach described in the previous section we have different values which have to be considered or adjusted to optimize the recognition of non-selfs in a decentralized system:

- The length of the self messages
- The receptor length of the antibodies
- The amount of different offsets among all antibodies where they start the comparison
- The choice and amount of antibodies distributed in the system and located at a node in the system

To evaluate the immune system approach we implemented the artificial immune system into our middleware  $OC\mu$  [16]. We also separated the key components from the system and built a simulation environment for faster evaluations. This simulation environment provides the possibility to generate a specific amount of selfs and antibodies with a given or calculated receptor length.

#### 3.1 The Length of the Messages



**Fig. 2.** Sections within some hashed self messages when a receptor length of six bits at offset 2 is used

Instead of grappling around with messages of different and maybe infinite length we transformed all the messages appearing in the system to a unified length by using a hash algorithm. Now the optimal length of the resulting hash value has to be determined and the most appropriate hash algorithm has to be chosen.

In our system we have  $S$  self messages. Antibodies always compare themselves to the hashed incoming messages at  $R$  contiguous bits, similar to [9], but always starting at a specific offset  $O$  (see figure 1) and always comparing the whole receptor.

We regard antibody-specific sections within the set of all self messages, where a section is defined as the part an antibody should compare to, i.e. the  $R$  contiguous bits of the antibody starting at its specific offset  $O$ . The sections of all hashed self messages with the same offset and length form a group (see figure 2). All the hashed self messages can be grouped in  $P$  groups - namely submessages of length  $R$  starting at position  $O$  from the hashed self messages. We only consider one of these groups because every group behaves identical. Thus one group consists of  $S$  messages of random bit sets of length  $R$  and the maximum amount of antibodies which can be created in this group is limited by the usage  $U$  of different bit sets in a group.

Because the system is aware of all the harmless self messages it is able to analyze the binary distribution within the different groups. The following pseudo algorithm shows how the possible antibodies can be created for a specific offset:

---

**Algorithm 1** Generating antibody receptors for a specific offset group

---

```

R = 1;
pattern = 0; // binary representation of the pattern
loop
  while pattern < 2R do
    if !groupMatches(pattern) then
      possibleAntibody(pattern); // negative selection
    end if
    pattern++;
  end while
  pattern=0;
  R++;
end loop

```

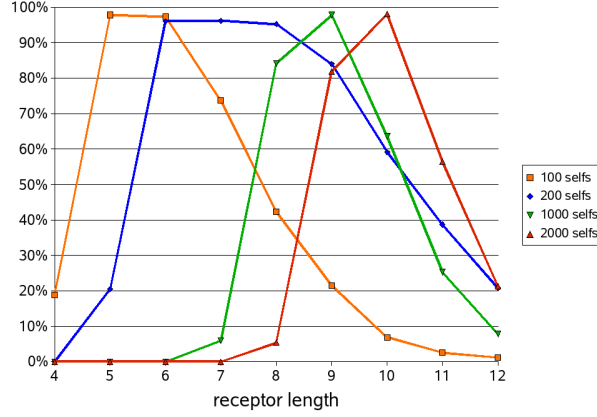
---

The algorithm produces all possible antibodies starting with the smallest receptor length of one bit. Because there is no upper bound for the receptor length the loop can be exited when enough receptors were created.

### 3.2 The Optimal Length of the Receptors

For determining the optimal length of the receptors we set up different simulator configurations with different amounts of self messages and tested the recognition of antibodies with different receptor lengths. The performance of the different types of antibodies is shown in figure 3 where always the maximum amount of antibodies were created in every test environment.

This experiment demonstrates that it is not efficient to choose antibodies with a fixed or even a random length for the receptors. Instead it shows that this value has always to be adjusted due to the amount of self messages known in the system. To achieve this we have to look a bit closer to some conditions. We consider a binary alphabet and hashed messages of length  $L$ . We use antibodies



**Fig. 3.** Recognition probability with different amount of selfs and receptor lengths

with a receptor length of  $R$  bits and those receptors can be compared to the hashed messages at most at  $P$  different starting positions or offsets:

$$P = 1 + L - R, \quad R \leq L \quad (1)$$

If every antibody has its fixed offset where it compares itself with the hashed message we end up in  $A_{all}$  different antibodies which match all possible selfs and all non-selfs:

$$A_{all} = 2^R \cdot P \quad (2)$$

Lets assume that we have  $S$  different self messages in our system which should not be recognized by the immune system. In the worst case we get an amount of  $A_{self}$  antibodies which should not leave the thymus because they match at least one self:

$$A_{self} = S \cdot P \quad (3)$$

This results in at least  $A_{eff}$  effective antibodies:

$$A_{eff} = A_{all} - A_{self} = 2^R \cdot P - S \cdot P, \quad R \leq L \quad (4)$$

The optimal length of the receptor can be determined when considering that the amount of antibodies should be equal or smaller than the amount of selfs.

$$A_{eff} \leq S \quad (5)$$

$$2^R \leq S \cdot \frac{1+P}{P} \quad (6)$$

$$R \lesssim \lfloor \log_2(S) \rfloor \quad (7)$$

This shows that the best length for the receptors should always be smaller than the binary logarithm of the amount of known self messages<sup>1</sup> but at least long enough to get a group usage below 100%. The group usage is the percentage value which embodies how many different bit patterns appear in that group with respect to all possible bit patterns.

Because the system is aware of all the harmless self messages it is able to analyze the binary distribution within the different groups. Thus a better length can be calculated because the mathematical calculation always assumes the mean probability which does not always match with the reality. We introduced a threshold for the group usage to calculate a good length for the receptors. Different tests showed that the recognition of non-selfs is best (with also low memory usage) if using a receptor length at which the group usage is slightly below 95%. In any other way there are too many antibodies to generate or not enough for a sufficient recognition rate.

### 3.3 The Amount of Offsets

In our research we tested different kinds of offsets which were used to match the antibodies with a hashed message to detect malicious objects. The easiest way is to define only a single offset and generate antibodies with receptors of length  $R$  which compare to the hashed message only at this specific offset. But this is exactly the same as if the hashed messages would have the same size than the receptors and the whole incoming messages always compares to the antibodies. This would result in a very bad recognition of non-selfs as the usage of this group corresponds to the probability for not recognizing the non-selfs.

In contrast in another test environment we designed the antibodies to use all possible offsets. This results in fully overlapping groups and thus less hashed messages can be generated that do not match any antibody. And as expected our tests showed that the more we overlapped the groups the better the recognition of non-self messages worked out. the best recognition was observed when using all possible offsets (from offset 0 to  $L - R$ ) and always the same amount of antibodies in each offset group.

### 3.4 The Right Choice of Antibodies

When not generating all possible antibodies, it is important how many antibodies to create and at which offsets. Because the system is aware of all selfs and thus it is aware of all used receptor patters in all groups we can generate the antibodies in a structured way. We tested three different methods for the creation of antibodies:

- Generate antibodies with fewest group usage first
- Generate antibodies with highest group usage first

---

<sup>1</sup> As the receptor length has to be whole-numbered we always use the next lower integer as result of the binary logarithm.

- Generate an equal amount of antibodies in every group

In our tests we set up a system containing 1000 self messages. Thus we generated different amounts of antibodies with the three methods mentioned above. The result of recognizing malicious messages is shown in figure 4. First we generated antibodies out of those groups which have the fewest usage but this resulted in a bad recognition of non-selfs – at least when generating only 1000 antibodies. A similar bad behavior was observed when choosing the second method where those antibodies were generated first which had the offsets of those groups with the highest usage of bit patterns. Apparently, the best method – also when generating only a few antibodies – is to generate an equal amount of antibodies in every group.

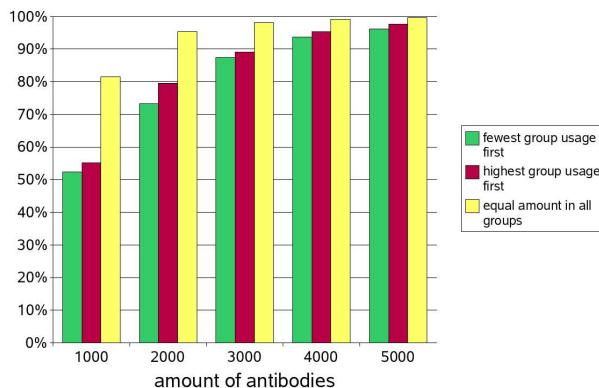


Fig. 4. Recognition with different methods of generating antibodies

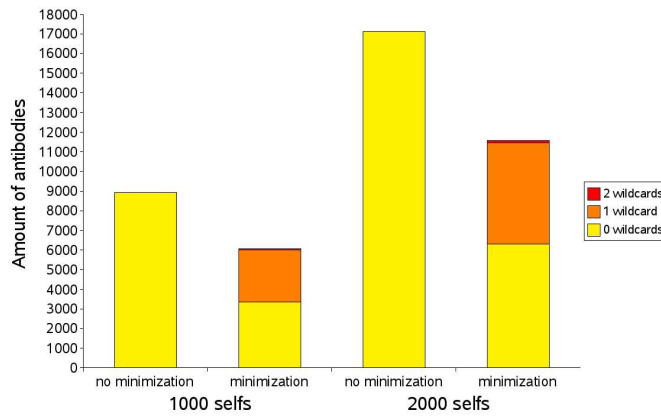
This experiment also showed that the recognition of non-selfs leads up to 99.3% when using 5000 antibodies of a receptor length of nine bits and equal distributed offsets among all antibodies in an environment of 1000 self messages.

## 4 Optimizations

### 4.1 Optimizing Memory Requirements by Merging Antibodies

At a first thought one would suggest to store the self messages instead of the antibodies and compare all incoming messages with the self set. This is correct when the system has only a few self messages but this research examines systems with high activity and thus many self messages. When looking closer to this issue the usage of antibodies should be preferred. In the system setup mentioned in section 3.4 it would need  $1000 \cdot 128bit = 15.6kb$  to store all known selfs. Compared to the amount of 5000 antibodies with a receptor length of nine bits and 128

different offsets this results in only  $5000 \cdot (9bit + 7bit) = 9.77kb$  memory to store these antibodies and still reach a recognition probability of 99,3%. The seven bits result from the storage of the specific offset which can take values from 0 to  $128 - R$ . Another benefit of choosing the antibodies is that only short receptors have to be compared instead of the whole hashed messages of 128 bits in length.



**Fig. 5.** Mimizing the amount of antibodies by merging at specific bits on the receptors

We propose a technique for memory requirement minimization which does not advance the recognition probability but minimizes the amount of antibodies to be stored in the system. Two antibodies with the same offset and with an identical receptor can be merged to one and the same antibody which is trivial. But also antibodies with the same offset and different receptors can be merged to one antibody if they differ at only one position on the receptors. In that case a new antibody can be created with a *wildcard* at this specific position which replaces the two antibodies. At compare time this wildcard position does not have to be compared because it treats both possible values – 0 and 1 – as a valid bit yielding even in a small speedup. Additionally antibodies with wildcards can be merged together when two receptors have the same wildcard positions and differ at exactly one further position.

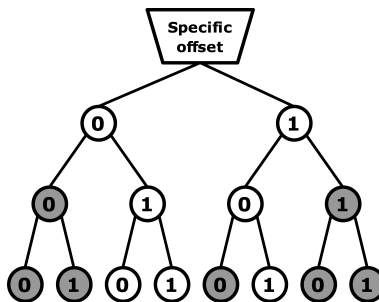
In our tests we were able to eliminate about 30% of the antibodies by merging them together. We evaluated the optimization with different amounts of randomly generated antibodies and figure 5 shows how many antibodies could be merged to antibodies with one and two wildcard positions when using a setup of 1000 and 2000 antibodies. Unfortunately in our test cases we were never able

to produce antibodies with three or more wildcards because this is very difficult when the group usage of selfs exceeds some specific value.

## 4.2 Optimizing Execution Time

In a human body the comparison of antibodies against appearing objects takes place in a three-dimensional environment and antibodies check themselves randomly all the time. Due to the one-dimensional architecture of computer memory we have to cope with some limitations in that aspect [2]. In our test cases we stored all antibodies in an array and thus the comparison against incoming messages takes some time - more precisely if we have  $A$  antibodies with a receptor length  $R$  we have to cope with a time complexity of  $O(A \cdot R)$  to compare an incoming message to all antibodies. That is not very efficient if we have a lot of different antibodies. Comparison would slow down the communication between applications.

To achieve a more efficient comparison a new storage mechanism has to be devised for the antibodies. Many antibodies correspond to other antibodies in specific parts of their receptor. So the idea grew to store the receptors in a binary tree. Thus the comparison can start at the root of the tree and follow the way down according to the binary pattern. If there is a complete way down to the bottom of the tree one antibody stored in that tree matched the incoming pattern (see figure 6).



**Fig. 6.** Representation of a receptor tree that contains the receptors 010, 011 and 101. White nodes represent enabled values at specific positions, gray nodes are values that are not available in any receptor.

This approach implements a separate perfect binary tree (without a root node) of depth  $R$  for each offset known in the system. A tree is analogous to the receptors of the antibodies at one specific offset if read from top to bottom and only traverses enabled nodes (enabled nodes are white colored in figure 6). Because every tree has to be perfect and every node consists of one boolean value a memory usage of  $(2^{R+1} - 2)$  bits is necessary for every tree with an offset in  $P$ . To compare incoming messages against the trees each tree has to be

passed according to the incoming pattern at the corresponding offset. If there is no enabled path from the root to the bottom of the tree the next tree comparison takes place which embodies the antibodies at another offset. If one tree matches at one specific path the message is treated as malicious. This comparison results in a time complexity of  $O(P \cdot R)$  and because in general  $P \ll A$  this approach is much faster than comparing all antibodies with a complexity of  $O(A \cdot R)$ . Further more  $P$  and  $R$  are fixed for a group of trees this results in  $O(1)$  for comparing incoming messages to all stored antibodies. In our tests the tree comparison resulted in a 30 times faster comparison than using the linear comparison algorithm.

## 5 Integration in the Organic Middleware $OC\mu$

The Organic Middleware  $OC\mu$  (formerly known as AMUN) [16] is a message-based middleware based on the peer-to-peer system JXTA (see figure 7). To realize the self-x properties from organic computing the middleware is extended by an organic manager and interfaces which add monitoring capabilities. Because of that manager the system is categorized as an organic middleware. Different services are started among the nodes of the system depending on a given configuration. Those services are not fixed to a node but can be relocated automatically by the middleware for self-optimizing the whole system if necessary. All services contain an unique ID and communicate through messages. All messages which are sent by a node always preambulate the event dispatcher where different message monitors can be placed either at the incoming or the outgoing interface. Message monitors can operate on incoming or outgoing messages without any control by the services (i.e. in this case a message monitor checks the messages if they are self or non-self). The goal of this work is to integrate the artificial immune system to prevent the system from malicious intrusive services or messages. The intrusion detection system in  $OC\mu$  currently consists of three components: The *thymus*, the *immune instance*, and the *intrusion detection monitor*.

### 5.1 The Thymus

The thymus is realized as a service in the middleware and is the cornerstone of the whole immune system. It has the ability to ask its local services for all message types known from the configuration and also to ask remote thymuses about the message types used on that node. This implicates that a thymus service also has the functionality to spread the known message types to other nodes as they are requested by remote thymuses. Another task of a thymus service is the proper generation of antibodies with respect to the systemwide known message types and to suggest a suitable receptor length of the antibodies to achieve an optimal recognition rate of intrusive message types. Generating antibodies is very time consuming. Thus the most thymuses in the middleware just send all their local message types to some dedicated thymus services which generate all necessary antibodies. Therefore every node in the middleware should run an instance of the thymus service - at least for sending the known selfs.

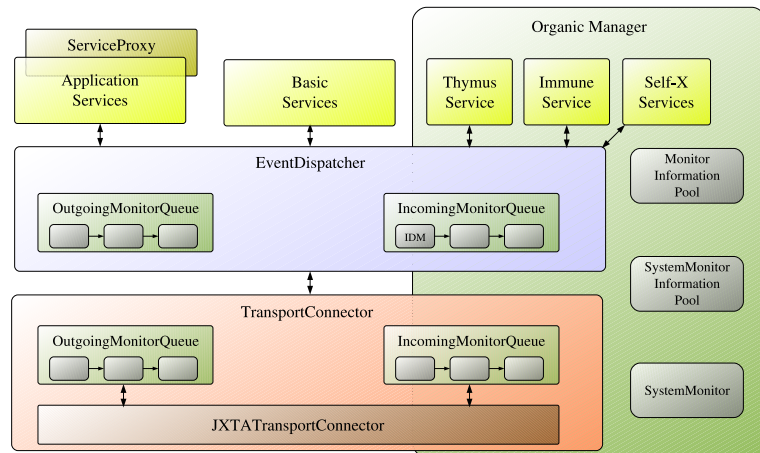


Fig. 7. The architecture of the organic middleware  $OC\mu$

## 5.2 The Immune Instance

The component called immune service embodies the instance which decides if some intruding message should be accepted or treated as non-self. Therefore this module is also realized as a middleware service. It is able to receive antibodies which are generated and spread by the dedicated thymuses around the middleware. Those antibodies are internally stored in an array for comparison to incoming message. In the next section we will show how to speed up the comparison when using a binary tree for storage.

## 5.3 The Intrusion Detection Monitor

The last component in this architecture is the intrusion detection monitor (IDM) which is placed in front of the incoming message monitor queue within the event dispatcher (see figure 7). Every incoming message is checked automatically for its validity. To realize this the monitor sends the message type and the ID of the sending service to the immune service. This service decides if the message either belongs to self and should be trusted or classifies it as non-self because it may be harmful to the middleware and its functionality. As for now the intrusion detection monitor only checks for malicious messages and can block them but it does not defend the intrusion yet. Tracking back a malicious message to its originator results in detection of a harmful service.

## 5.4 Efficiency of the Artificial Immune System

There are two issues which have to be considered for evaluating the efficiency. On the one hand there is the time needed for generating the antibodies and on the other hand the time needed to check an incoming message. Generating

antibodies is time consuming anyway and this work is only done on dedicated nodes. Due to that fact the job of the thymus will not be measured. Every incoming message on a specific node has to be categorized as good or bad. This check might also be time consuming and may affect the response time of the system. Due to that issue we measured the delay which appears when sending all messages through the monitor. When the detection monitor is enabled and with the optimizations described in the previous section, a node in the middleware only needed about 0.3% to 1.3% more time to process any messages than the normal message handling in OC $\mu$  without any checks. These values were measured with 5000 selfs, about 15,000 antibodies and 10,000 randomly generated messages in a running OC $\mu$  system.

## 6 Related Work

Non-immunological detection systems are mostly signature based or at least they use the information gained from previously detected anomalies. For example current virus scanners contain signatures from known viruses and scan files for a match with those signatures. But there are also approaches which can detect unknown or new intrusions because of not comparing to known patterns but the other way round – using the negative or clonal selection.

Hofmeyr and Forrest developed an architecture for an artificial immune system to detect intrusions in a network [9]. They also modeled their immune system on the biological immune system described in section 2. The antibodies and the messages in their system consist of bit strings with a fixed length and the system compares the messages appearing in the network to all created antibodies. If  $r$  contiguous bits are identical the message is treated as non-self otherwise the message is accepted by the immune system. The number  $r$  has to be adjusted in their system to tune the recognition probability and can have values from one bit to the entire length of the messages [10]. Our approach follows a similar way to design the antibodies but shows the basic correlation between the amount of self-messages and the length of the receptors needed to get an optimal recognition probability in any domain.

A further approach was made at the Swiss Federal Institute of Technology in Lausanne where an artificial immune system was proposed which can detect misbehavior in mobile ad-hoc networks because the dynamic source routing in those networks can be used by malicious nodes to vulnerate the system. They generated random antibodies using negative and clonal selection. With clonal selection antibodies generate copies of themselves with similar but not strictly identical receptors [1]. Our immune system aims at another application domain, i.e. to protect our organic middleware from intrusive services. Plus, in our artificial immune system we are aware of all selfs used in the system and thus we have the ability to know what belongs to self and non-self.

## 7 Conclusions and Future Work

The tests showed that the recognition rate was very good (up to 99.3% in some test cases) when choosing a suitable receptor length and an appropriate amount of antibodies. We also looked closer at the speed of detecting selfs and non-selfs and found a way to speed up the comparison enormously by using a binary tree.

The organic middleware [16] is a message oriented middleware based on a peer-to-peer system. To realize the self-x properties from organic computing paradigms the middleware is extended by an organic manager and interfaces which add monitoring capabilities. As for now we implemented the automatic generation and distribution of the antibodies in the middleware environment and we also proposed some improvements and optimizations which lead to lower memory usage and a faster detection.

In a next step we integrated a reaction system around the detection mechanism at an early stage. Thus the middleware will be able to deactivate malicious services or shut down infected nodes in the network to prevent other nodes from the intrusion. This leads to a so-called self-healing mechanism which recovers the system after a successful defeating of malicious events.

## References

1. Jean-Yves Le Boudec and Slaviša Sarafijanović. An Artificial Immune System Approach to Misbehavior Detection in Mobile Ad-hoc Networks. In *In Proceedings of Bio-ADIT – The First International Workshop on Biologically Inspired Approaches to Advanced Information Technology*, pages 96–111, Lausanne, Switzerland, January 2004.
2. Mark Burgess. Computer Immunology. In *Twelfth Systems Administration Conference (LISA '98)*, Boston, Massachusetts, December 1998.
3. Robert M. Corless, David J. Jeffrey, and Donald E. Knuth. A Sequence of Series for the Lambert W Function. In *International Symposium on Symbolic and Algebraic Computation*, pages 197–204, Maui, Hawaii, USA, 1997. ACM.
4. L. N. de Castro and J. Timmis. Artificial Immune Systems: A Novel Paradigm to Pattern Recognition. In J. M. Corchado, L. Alonso, and C. Fyfe, editors, *Artificial Neural Networks in Pattern Recognition*, pages 67–84, SOCO-2002, University of Paisley, UK, 2002.
5. Leandro N. de Castro and Fernando J. von Zuben. *Biologically Inspired Computing*. Idea Group Publishing, 2005.
6. Patrik D'haeseleer. An Immunological Approach to Change Detection: Theoretical Results. In *9th IEEE Computer Security Foundations Workshop*, Dromquinna Manor, County Kerry, Ireland, June 1996. IEEE.
7. John M. Hall and Deborah A. Frincke. An Architecture for Intrusion Detection Modeled After the Human Immune System. In *Proceedings of the International Conference on Computer, Communication and Control Technologies*, volume 6, pages 75–78, 2003.
8. Emma Hart and Jonathan Timmis. Application Areas of AIS: The Past, The Present and The Future. In *4th International Conference on Artificial Immune Systems (ICARIS 2005)*, volume LNCS, pages 483–497. Springer-Verlag, 2005.

9. Steven A. Hofmeyr and Stephanie Forrest. Architecture for an Artificial Immune System. In *Evolutionary Computation* 8, number 4, pages 45–68, Massachusetts Institute of Technology, 2000.
10. Steven Andrew Hofmeyr. *An Immunological Model of Distributed Detection and Its Application to Computer Security*. PhD thesis, University of New Mexico, May 1999.
11. Zhou Ji and Dipankar Dasgupta. Estimating the Detector Coverage in a Negative Selection Algorithm. In *Genetic and Evolutionary Computation Conference (GECCO 2005)*, pages 281–288, Washington DC, June 2005. ACM.
12. Jeffrey O. Kephart and David M. Chess. The Vision of Autonomic Computing. *IEEE Computer Society*, pages 41–50, January 2003.
13. Christian Müller-Schloer. Organic Computing Initiative. Published as PDF on <http://www.informatik.uni-augsburg.de/lehrstuehle/sik/research/organiccomputing/download/OC-english.pdf>, April 2004.
14. Ronald Rivest. The MD5 Message-Digest Algorithm. Technical Report Request for Comments: 1321, Internet Engineering Task Force (IETF), April 1992.
15. Anil Somayaji, Steven Hofmeyr, and Stefanie Forrest. Principles of a Computer Immune System. In *New Security Paradigms Workshop*, pages 75–82, Cumbria, UK, 1997. ACM.
16. Wolfgang Trumler, Faruk Bagci, Jan Petzold, and Theo Ungerer. AMUN - autonomic middleware for ubiquitous environments applied to the smart doorplate. *Advanced Engineering Informatics*, (19):243–252, April 2005.