

Self-Aware Software – Will It Become a Reality?

Peter Andras¹ and Bruce G Charlton²

¹ University of Newcastle Upon Tyne,
School of Computing Science,
Newcastle upon Tyne, NE1 7RU, UK
`peter.andras@ncl.ac.uk`

² University of Newcastle Upon Tyne,
School of Biology(Psychology),
Newcastle upon Tyne, NE1 7RU, UK
`bruce.charlton@ncl.ac.uk`

Abstract. The possibility of building self-aware software fascinated computer scientist since the beginning of computer science. Research in AI, and in particular on software agents, agent system, computational reflection and reflective software delivered interesting results which moved towards the development of software systems with features of self-awareness. However, these approaches have not so far generated any clear success in terms of real and useful self-aware software. Here we introduce the theory of abstract communication systems, which describes the world in terms of systems and their environment. Systems comprise dense, inter-referencing clusters of communications. We analyse natural self-aware systems highlighting the critical features which make them able to be self-aware. We analyse software systems in terms of abstract communication systems theory and compare their critical features with these natural self-aware systems. We describe the necessary features of hypothetical self-aware software, discuss the existing barriers that stand in the way of realization of such systems and how these might be overcome.

1 Introduction

The concept of self-aware software appeared very early after the building of first large programmable computers and provided the foundation for science fiction novels and films (e.g., 2001: A Space Odyssey). In scientific terms these ideas led to the emergence of the domain of artificial intelligence as part of computer science [11], [17]. An early failed attempt to build software with self-awareness used perceptron neural networks, which were believed to lead to the emergence of some kind of consciousness (e.g., [57]). Other attempts based on symbolic reasoning also led to failures in terms of producing truly self-aware software systems [13], [49].

The idea of building an adaptive self-aware software system that is able to sense itself and maintain itself producing appropriate adaptive behaviours (dependent on the environmental stimuli and on the state of the system) appears

perpetually fascinating for many computer scientists (e.g., [23], [28], [37], [42], [54], [63], [64]). A direction that grew out from classical artificial intelligence research is focusing on intelligent agents and multi-agent systems (e.g., [2], [31]). The aim of this research is to build software agents that can behave intelligently in a limited sense (e.g., within a strictly defined knowledge domain, like collecting, selecting, categorizing and packaging for delivery of financial news items) and which can support the professional activity of their owner. One step further the aim is to build systems of such agents, which are able to perform meaningful negotiations over exchanging services, representing the interests of their owners [2].

Another important direction is the research on computational reflection [36], [42], [51] and on reflective software systems [14], [18], [21], [52]. Reflective software systems aim to be able to evaluate themselves and generate adaptive responses to external stimuli, such that the response is appropriate for the state of the software system. Reflective software research also feeds into the research on intelligent agents, by providing core software for some of these agents [62]. A more application oriented related area is the research on self-monitoring and self-healing software [6]. The aim of this research is to build software systems that are able to evaluate their own state, detect faults and errors and automatically correct them returning the system to a 'healthy' state. Recent works on aspect-oriented software development [19], [33], [54] are very promising in terms of building self-monitoring applications, which is an important step towards self-aware software systems.

The above mentioned works on building software systems that show some level of self-awareness in an adaptive and responsive manner led to limited success. It appears that the most critical barrier that these systems cannot overcome yet is that of generating adaptive novel responses to previously unseen situations. In other words these systems are able to do what they were programmed for and to make the decisions that their programmers considered when they programmed them, but they cannot surpass these pre-programmed behaviours by generating new ones, which were not hard-coded by their programmers. In our view a major reason behind this lack of progress is that all these systems are designed to perform an externally implied function in some kind of efficient manner. This approach implies that there is no 'self' in a real sense of the software system from the point of the view of its design, and that the resources of the system are usually optimized sufficiently to prevent the emergence of suboptimal behaviours (these are usually considered erroneous ones), which otherwise is the usual way of emergence of new behaviours in the case of natural self-aware systems (e.g., bacteria, plants, animals, social organizations) [3], [16], [41], [48].

We note that there are positive signs towards software development environments, which may allow the emergence of the 'self' of software systems. Languages like Java and Smalltalk which implement reflection allow some level of structural self-inspection and self-modification (i.e., structural reification) [22], [28], [51], [61]. Component-based (e.g., JavaBeans, web services) [30] and aspect-oriented programming [19], [33], [54] may represent a way away from strict

functional optimality, which in our view prevents the emergence of self-aware software.

The abstract communication systems approach offers a new way to look at software systems, and in particular offers an insider view of these systems. The theory of abstract communication systems [4], [15], [16], [41] was developed originally in the context of social systems by Luhmann (1996), having its roots in works on autopoietic systems [45] and in works on decision making in organizations [10], [58], [44].

In the interpretation followed here, the abstract communication systems approach views systems as dense collections of inter-referencing communications generated by communication units, which themselves are not part of the system. Such communication systems can be used to describe biological and social systems providing revelatory insights into the nature of these systems. We propose here to use this approach to analyse software systems and highlight the requirements for the building of self-aware software systems.

First, we provide a brief overview of the theory of abstract communication systems, explaining key concepts and providing examples that highlight key features of the explained concepts. Next, we discuss two natural self-aware systems, living cell and social organizations, emphasizing their critical components that provide the foundation for their self-awareness. This is followed by the analysis of software systems using concepts of abstract communication systems theory, focusing on components of software systems that are required for self-awareness in our view. Finally, the paper is closed by a discussion about how realizable we see the generation of self-aware software systems considering the requirements for this discussed previously.

2 Abstract Communication Systems

In this section we introduce fundamental concepts of abstract communication systems theory following the work of Luhmann (1996) and our earlier works [4], [15], [16], [5]. Each introduced concept is explained in theoretical terms supported by practical examples highlighting the relevant features of the concept.

Communications and Communication Systems. Communications are sequences of symbols communicated between communication units. Abstract communication systems are made of such communications between communication units. The communication units are not part of the system, since they are not themselves communications but instead transmit and receive communications. Communications reference other communications, in the sense that the sequence of symbols contained in a communication is dependent on the contents of other earlier or simultaneous communications and thereby refer to them. A dense cluster of inter-referencing communications surrounded by rare set of communications constitutes a communication system. In quantifiable terms it may be said that a system is a 'significantly' dense concentration of inter-referenced communications which persists over a 'significant' timescale - in which the cut-off levels of significance define the probability that there is indeed a system.

For example the system of computer science contains all communications which reference earlier scientific communications from the domain of computer science and which follow the rules of these scientific communications (e.g., allowing the possibility of falsification, using logical reasoning, discussing admissible topics using admissible arguments etc.). A large part of these computer science communications are scientific papers, which explicitly reference other scientific papers, and use the conclusions of earlier papers as premises of the logical reasoning presented in the paper. According to systems theory, the human computer scientists are not part of the system of computer science, only their scientific communications about computer science topics are part of this system.

A communication system is defined by the regularities that specify how referenced communications determine the content of a referencing communication. All communications that follow the set of rules defining the system are part of the system. Other communications that do not follow the rules of the system are part of the system's environment. Therefore from the systems perspective the world is constituted by the system under consideration and its environment - and there are as many such 'worlds' as there are systems, such that the same communication will have different meanings in different systems or be included in one system but not another. The set of regularities of referencing constitutes an abstract grammar, which defines an abstract language, characteristic of the system. For example the sciences of economics and medicine have different specialist languages, and scientific communications belong to one of these sciences according to whether they follow the rules of the specific language.

Systems Are Self-Reproducing. Communication systems reproduce themselves by recruiting new communications, which follow the referencing rules of the system. How successful the recruitment of new communications is, depends on earlier communications generated by the system and on the match between the system and its environment. We can view the system as a self-describing system made of communications, which at the same time describes its environment in a complementary sense. (In other words, the system's only knowledge of its environment is within the system itself - the system models the environment, and that model is the sum of its knowledge of the environment.) Better - ie. more complex and adaptive - descriptions of the systems environment potentially lead to higher success in recruiting new communications and more rapid reproduction and expansion of the system.

For example we may consider the case of artificial intelligence of the 1950s-1960s. During this period AI research used relatively simple assumptions about natural intelligent systems (e.g., relatively simple artificial neural networks were supposed to simulate biological neural networks) and made high promises. AI research expanded fast in this period. In a relatively short term the promises made were proved to be unachievable (e.g., the works of Minsky and Papert on perceptrons). This led to a massive reduction of funding for AI research and the shrinking of the volume of science communications in the area of AI. After the 1980s revival of AI research science communications in this domain developed better descriptions of the environment, which led to fewer and smaller failures

and supported the recent expansion in the domain of novel AI (e.g., data mining, intelligent control). Better descriptions of the system environment in case of recent AI fuel the expansion of the system of this science, while the not so good environment descriptions of classical AI led to the shrinking of AI during the 1970s.

The system communications are about the system itself. Taking another approach, the system communications reference other system communications in order to prove that they are part of the system (i.e., that they are correct according to the rules of the system). If the communications lead to continuation the process of proving that they are correct continues. If the system is able to exist, i.e., to generate/recruit new communications according to the rules of the system, this implies that the proving process of the correctness of earlier communications continues. In general it is not possible to prove the correctness of system communications; it is possible to prove only the incorrectness of them, when there is not further continuation of communications rooted from the original communication. We call this the Popper Principle, i.e., that only the falsity of system communication can be proven by stopping the generation of communications rooted from the communication in question.

More Adaptive Systems Out-Compete Less Adaptive Ones. Systems that reproduce and expand faster than other systems may drive to extinction the slower reproducing and expanding systems. The limits of system expansion are determined by the probabilistic nature of referencing rules. A communication may reference several earlier communications indirectly through other referenced communications constituting referencing sequences of communications. The indeterminacies of referencing rules determine how long can be such referencing sequences of communications before the later communications become a random continuation. Longer referencing sequences of communications (i.e., more detailed descriptions) allow better descriptions of the systems and its environment. The optimal size of the system (i.e., the number of simultaneous communications being part of the system) is also determined by the indeterminacies of referencing rules. Systems that overgrow their optimal size may split into similar systems.

For example we may consider the introduction of electronic storage and management of information in companies. Before this, information was mainly stored on paper. Paper storage of accounting data for example increases the likelihood of making errors in calculations compared to electronic handling of the same data, and also makes it difficult to handle very large amounts of data. Electronic storage and data management decreases the likelihood of calculation errors and allows efficient organisation and handling of huge amounts of data. In both cases the data describes the environment of the company, but in the case of electronic data it is possible to reliably perform much more complicated operations with the data (i.e., longer sequences of such operations) than in the case of paper based data. This implies according to systems theory that the environment descriptions of companies using electronic data are better than the environment

descriptions of companies using paper based data. Indeed, companies adopting electronic data easily out- compete companies using paper based data.

Systems Increase Complexity by Developing Subsystems. Communication systems may develop subsystems that are systems within the system, i.e., they constitute a denser inter-referencing cluster within the dense communication cluster of the system. Communications that are part of subsystems follow system rules with additional constraints that are characteristic of the subsystem. More constrained referencing rules decrease indeterminacies and allow the system to generate better complementary descriptions of the environment and expand itself faster than systems without subsystems. Systems may also change by simplification of the set of their communication symbols (i.e., reduction of the number of such symbols). This may lead to reduction of indeterminacies in the referencing rules. Consequently systems with simpler sets of communication symbols may expand faster than systems with larger sets of communication symbols.

For example we may consider early computer software systems that aimed to deal with a wide range of problems and had a monolithic architecture. Extending such software systems was difficult, and led very fast into the generation of inconsistent data and conflicts between components of the system. More recent software systems are modular, having specialized parts dealing with specific classes of problems. Expanding these software systems is much easier, and the likelihood of running into major integration problems is relatively low. Recent advances in the area of software development led to the building of template libraries and design patterns, which simplify the building of software systems, providing standardised building blocks for them. Due to the availability of such standard components large software systems can be developed and upgraded faster than before, and current software systems grow much larger than any previous software system.

Systems with Memory. Another way of extending reliable descriptions of the environment (i.e., non-random sequences of referencing communications) is by retaining records of earlier communications, i.e., by having memories of earlier communications that can be referenced by later communications. In a sense we can view such memories as the creation of new communication units (or recruitment of communication units) that produce for a certain period a certain communication that can be referenced in place of some other communication (i.e., the one which is represented by the memory). Having memories reduces the indeterminacies in referencing by allowing direct referencing of much earlier communications, instead of referencing them through a chain of references.

An alternative way of considering memory systems is that they are communication systems that 1. model the communications of the main system and 2. communicate using longer lasting communications than the main system. This means that the communications in the memory system provide a longer-lasting record of events in the main system.

However, in strict systems theory terms, this is not quite accurate - since the 'memory function' of a system only refers to its function as perceived by another system. Furthermore, the 'memory function' only refers to the memory systems communications with its environment, yet by definition a complex system has a much denser communication referenced to itself than its communications with its environment.

The implication is that a memory system is first and foremost a densely self-referencing complex system with the implicit function of self-reproduction and growth in complexity by means of modelling its environment. Therefore, memory systems arise in the context of an environment, differentiate and expand due to their self-reproductive qualities; from this it follows that the internal evaluation processes of memory systems are not primarily concerned with providing an accurate data base for other systems.

What makes such a system able to function as a memory for another system is firstly that the memory system has longer lasting communications than the system using it as memory, and secondly that the memory system's model of its environment is under selection by the system which is using it as memory. This implies that the memory systems model of another system using it as memory is not a representation of the other system, but is a model which has been selected by that other system.

This illustrates that memory systems communications are (like all systems) mainly internal, and the vast majority of the communications of history have nothing directly to do any environmental system. But the environmental systems usage of some of the communications of memory systems will amplify some parts of the memory system and suppress others, reinforcing some random changes in the evaluation criteria of memory systems and suppressing others such that the memory system will evolve, will grow in complexity in particular ways, and to an external observer can be seen to perform its 'memory function' more efficiently.

Consequently, systems with memory can expand faster than systems without memory. In the context of human communications such memories of communications are written or otherwise recorded verbal communications and human artefacts that can be seen as memories of human behavioural, verbal and written communications that led to the creation of the artefact.

As an example we may consider the effect of printing on science. Before printing was invented science developed slowly, as it was based on difficult and time consuming reproduction of scientific texts by handwriting. After the invention of printing the system of science was able to expand much faster than before, having more available memory communications (written texts) because each individual written scientific communication is longer-lasting than verbal communications- especially when these written communications are incorporated into a specifically archival system which generates new forms of complex communication based around these long lasting communications (i.e., science librarianship).

Identity-Checking Subsystems. Systems with memory may develop an information subsystem (the memory is information about the past of the system)

consisting of communications between communication units generating memory communications.

Information system communications reference memory communications and can be seen as representations of information processing operations - so that an information system may be defined as communications about memory communications. Hence, the discipline of history may be considered a type of information system, since history consists of communications about communications concerning the past.

Memories of information systems communications are referenced in future information system communications and provide the blueprint for processing of memories. Information system communication memories last longer than memories of other communications, and constitute long-term memories of the system, while memories of other communications are the short-term memories of the system. The information subsystem emerges if information processing communications constitute a dense cluster of inter-referencing communications determined by a set of characteristic referencing rules. Having an information subsystem allows combination of memories and by this the generation of descriptions of the environment which are better than such descriptions in systems with memory but without information subsystem.

But the information system is not primarily concerned with functioning as a memory system - its memory function is an interpretation reached by an external observer analysing the relationship between two systems, and concluding that one system is functioning as memory for another. For example, the system of the discipline of history (as exemplified by the written communications of professional historians) is - from the perspective of historians, its own justification - its implicit aim being to do ever more history. But from the perspective of the political system, the function of the discipline of history is (approximately) to be a repository of information about the past of the society which can be used for political purposes (ie. for getting and retaining political power). In modern democratic societies political and other social systems (such as the legal system, science, education and the mass media) combine to exert a selection pressure on the discipline of history such that the evaluation criteria of history become 'scientific' - in other words history is meant to be true and internally consistent, as a basis for understanding the past and planning the future for many social systems. But in traditional or 'totalitarian' societies, the political/ military/ religious ruling system exerts a monolithic selection pressure on the discipline of history, which evolves to have a quite different (and non-scientific) function of justifying the perpetuation in power of the ruling system.

We may consider as another example the solving of a software development problem by having a series of ad-hoc meetings. The participants of the meetings may remember ideas discussed at previous meeting and occasionally may reference them, but it will be very difficult to arrive to a reasonably good solution without systematic analysis of earlier ideas and developing elaborations of these ideas. Finding the desired solution will happen much faster if the meetings are minuted, analysis and synthesis tasks are assigned to participants, and commu-

nications about memories of earlier communications are generated in form of reports that are referenced during the next and later meetings. In this way the group of developers will progress relatively fast towards an acceptable solution and the system of the software developer company will expand faster.

The information subsystem of a system provides the references for identity checking communications of the system. The blueprints of information processing held in long-term memories of the system are referenced by identity check communications and allow checking the validity of communications within the context of the system. Having an information subsystem that provides references for identity check communications decreases the likelihood of generating wrong communications that cannot be referenced according to the rules of the system. Reducing the likelihood of wrong communications helps the expansion of the system by providing guarantees that the system communications are correct and can be referenced by further system communications. The identity checking communications turn the information processing blueprints into a self-model of the system. The self-model of the system is a simplified representation of the system and of its environment in a complementary sense, containing the information processing rules of the system.

For example we may consider a small start-up company having the founders and a couple of other people as employee. The company runs mostly in an informal manner, although standard record keeping is already in place and the company has its statutes regulating the structure of the company (which is almost invisible during the everyday work), decision rights of shareholders, etc. As the company grows and gets larger contracts and hires more employees it turns into an established company, with specialised departments (e.g., sales, HR, development), sets of rules that describe how information is processed within the company, and an elaborate visible multilevel structure (e.g., offices indicating status within the company). The company builds an extensive self-model and an elaborate identity checking system based on its information system communications. Identity check communications guarantee that within company processes and services delivered outside of the company meet the expected quality standards and together with the self-model of the company provide the foundation for the building of the company identity. The company would not be able to meet its growing commitments without this transformation. Having in place the information subsystem, the identity checking communications and the self-model allows the company to grow fast and conquer its market.

Faulty Communications. Faulty communications may occur in systems. Faulty communications are defined as those which do not fit the lexicon of the system's language or, which have zero likelihood to be produced in a certain context according to the rules of the system's grammar.

For example in case of human speech the pronunciation of meaningless words (a sequence of phonemes not associated with any word of the lexicon of the language of the speaker), such words are faulty communications. To differentiate between faulty communications of the system and communications, which are outside of the system (although produced by communication units, which

produce communications that are part of the system), we need to consider the referencing set of the communication. If the referencing set contains exclusively or dominantly system communications, and the communication in question is produced instead of a regular system communication that should follow by application of some system rules, the communication in question is a faulty communication.

For example, if the meaningless word occurs during meaningful speech, we have a faulty communication, while if a human produces a meaningless 'word' by the understanding of another human, this may not be the case of faulty communication (e.g., if somebody speaks Chinese words in an English environment, without the intention of linking these communications to other English communications).

In some cases faulty communications do not lead to any continuation communication within the system (e.g., communications outside of the lexicon of the system). In other cases system communications may reference faulty communications. Such cases may lead to further problems within the system and may cause the system to fail. However this is not necessarily the case - for example evolution by natural selection proceeds by genetic mutations which are faulty communications. Most genetic mutations are fatal to survival, or deleterious and damage the organism, however a minority of mutations are 'mis-interpreted' by the system as being meaningful, and by chance these mutations improve the adaptiveness of the organism enabling it to reproduce relatively more effectively than the organisms without the mutation - hence the system constituted by organisms of the same species can grow in complexity.

Communication Errors. Errors of communication systems are defined as cases when communications happen according to the rules of the system, but the system of communications cannot lead to continuation because of environmental constraints. In other words, communication units that are expected to produce the continuation communication are unable to do this, due to their participation in other environmental communications - for instance when an organism with no visual system is desiccated by sunlight since it cannot detect or respond to the light. The death of the organism is not due to a fault in the system communications, but to the error constituted by limitations of the organisms system of communications.

Errors therefore mean that the system's description of the environment is not correct in the specific circumstance being experienced. Of course, all system descriptions of the environment are necessarily incorrect in an absolute sense because the environment is more complex than the system. However there is no error so long as the environment description is 'good enough' to enable the system communications to continue. An error is said to occur when the incorrectness of environmental description leads to an actual failure in continuation of communications.

Continuations of faulty communications may lead to errors in the above sense. If there is no continuation of the faulty communication, the error occurs there, as there is no continuation of the system communications. If there are continuations

of the faulty communication, these continuations aim to decide whether the faulty communication is part of the system or not. Such communications may reach the 'not part of the system' decision according to the rules of the system, with the consequence of ending the sequence of communications by a terminating communication, implying no continuation within the system. If the halting of continuation communications happens when there are communications supposed to be the continuation of the most recent communications, the system reaches an error in the above defined sense.

Errors may occur even when there is no faulty communication at the root. Occurrence of such system errors are signs of the mismatch between the system's description of the environment and the actual environment. Mismatch errors imply that some of the rules defining the system are pragmatically wrong (i.e., they do not fit the environment as it is being experienced). The system's action on finding errors is the generation of communications that check the validity of communications that led to the communication triggering the error. These checks aim to find the root of the error, and terminate the continuations of communications branching out from the root of the error.

For example, in natural sciences hypotheses and theories are built and incorporated into ongoing scientific work. But when the experimental results 'falsify' the theories by not confirming their predictions, this prevents continuations of communications until the science is revised to remedy this. The root of the wrong theory is invalidated, together with scientific communications branching from this root, and these are eliminated from that science so that communications can continue. This may be described as 'purging' the system after an error to excise the error and its continuations.

Purging the system after an error may have very severe implications for the system. The invalidation of the branches emerging from the root of the error will probably shrink the system considerably in the short term. If large scale shrinking of the system happens the system encounters a failure. System failure may even lead to the dissolution of the system. For example, the experimental findings invalidating the roots of alchemy theories led to the elimination of the alchemy from the system of sciences. In case of the Enron company (an US electricity distribution giant until 2000), the finding that the roots of the accounting of the company were false led to the rapid collapse of the company. The attempts to purge the errors from the Russian economic system caused a collapse in economic output which has lasted 15 years, so far.

Self-Awareness in Systems. Abstract communication systems are self-aware systems if they have an information subsystem which generates an adaptive self-model of the system providing reference for identity check communications. In other words, self awareness implies the evolution of an information sub-system in the first place, and evolution of particular properties of this information sub-system. It is important to recognize that, like all systems, the internal communications of the information subsystem is differentially much greater than the external communications, and this differential increases as the complexity of the information subsystem increases. This implies that the function of self-awareness

(like all memory functions) represents only a minority of the communications of the system which enables self-awareness.

Consequently, self-aware systems generate adaptive actions and form adaptive perceptions representing the adaptive changes in their self-model. The adaptive changes in the self-model happen in response to faults, errors and failures experienced by the system. Purging the system of communications leading to errors imply imposing new structures and possibly new information processing rules. Simplification of communications, emergence of subsystems and interactions with other systems may also imply changes in the information processing rules. Changes in the rules of information processing are reflected in changes of long-term memories and corresponding changes in the self-model of the system.

For example a company may experience difficulties in selling their products in their market. These difficulties are errors within the system, as expected continuation communications (i.e., selling of products) do not follow previous communications (i.e., those which lead to the production and packaging for selling of the products). In response the company may revise its own regulations and other communications to identify the roots of the problem. If it is found that the company's technology is unable to produce sufficiently competitive products in some market of the company the company may change its technology or may turn its resources and technologies to produce different products. This change happens by modifying the regulations of the company and by adapting the self-model of the company. After the adaptive change of the self-model of the company identity check communications will change and the company will generate new adaptively changed actions and experience adaptively changed perceptions. In some cases the company may survive for very long time, while moving from one market to another, adaptively changing its self-model and identity in order to fit better to its environment (e.g., the BARCO company produces visualisation and optical monitoring equipment today, while originally it was founded as the Belgian - American Radio Company, which produced radios and later TVs).

3 Memory and Information Subsystems

We discuss in this section two natural systems, which feature self-awareness. The first system that we discuss is the cell which is a biological self-aware system producing adaptive responses to external stimuli in function of the state of the system. The second system that we discuss is the system of a human organization (e.g., a company or government department), which is again a self-aware system (i.e., a type of 'management'), which senses itself and produces adaptive and possibly innovative responses depending on external stimuli and the state of the system.

The critical features of the discussed systems for the presence of self-awareness are that they possess both short- and long-term memories and an information subsystem, which processes and creates new memories, and that they have an adaptive self-model that is referenced by their identity check communications. ('Adaptive' means that the self-model has evolved under selection

pressure from the system for which it serves the self-awareness function.). The memories are produced by communication units, which reproduce earlier communications that happened within the system. The information subsystem is a part of the communication system which generates new communications about memory communications (i.e., by referencing memory communications) and leads to the generation of long-term memory communications representing processes of combinations and derivations of existing memory communications.

3.1 Cells: Proteins, RNA and DNA

Living cells can be analysed in phenomenological terms by listing their components and the properties and behaviours of these components. This traditional approach is followed by most biology books, describing cells as a complex machinery made of cell membrane, cytoplasm, ribosomes, mitochondria, chloroplasts, Golgi organelle, cilia, flagella, centrosome, lysosomes, endoplasmic reticulum, nucleus and possibly various other cellular organelles [48]. For each of these components their structure and behaviour can be described, listing proteins, lipids and other molecules that compose them, and describing changes of them in response to various stimuli (e.g., pump molecules residing in the cellular membrane may introduce or expulse some ions or smaller molecules into/from the intracellular fluid).

An alternative way of looking at cells is to consider them as abstract communication systems, in which proteins and other molecules are the communication units and their interactions are the communications. In this sense the cell is identified as the set of interactions between proteins and other molecules (e.g., ATP, RNA) (Andras and Andras in press). All cellular components can be seen as well organized spatio-temporal patterns of interactions between such molecules, the proteins playing a central role in most of these interactions. Molecular interactions reference earlier interactions in the sense that the participating molecules are results of earlier interactions between molecules (e.g., proteins form intermediary complexes, which lead to new molecules, including proteins and possibly now conformations of the participating proteins, the resulting proteins participate in new molecular interactions, referencing earlier interactions which led to their formation or transformation). Such inter-referencing interactions form a dense cluster within the cell, surrounded by relatively rare interactions with molecules, which are outside of the cell. The density boundary of the cell system is materialized as the cell membrane.

Memories of interactions between proteins are contained in RNA molecules. The appropriate decoding of RNA molecules into proteins at the ribosomes guarantees that the appropriate types of proteins are available within the cell and that the appropriate molecular interactions happen within the cell delivering the expected functionality of cellular organelles into which the produced proteins are incorporated. The RNA molecules may change during evolution. Frequent or important interactions between proteins leading to stable protein complexes may become encoded by a single RNA, coding directly for protein complex emerging from the interactions of proteins. The RNA molecules produce the primary memory subsystem of the cell, allowing the reproduction of earlier communications

(i.e., molecular interactions) by allowing the reproduction of proteins. The RNA memory system is a short-term memory system, the RNA molecules being able to exist for a relatively limited time period (e.g., half-life of mRNA molecules is around 6 hours [55]).

The short-term memory molecules of the cell participate in many interactions between such memory molecules (i.e., RNAs). Well known and newly discovered RNA molecules, like mRNA, rRNA, tRNA, siRNA [1], [47], microRNA [39] and others interact with each other regulating the translation of memories encoded in mRNA molecules into proteins. The result of these regulatory interactions is the appropriate production of types and quantities of proteins within the cell. The interactions between RNA molecules rearrange some of them (e.g., splicing of immature mRNA) and produce RNAs corresponding to the right sequence of proteins that need to be produced or RNAs which can communicate with others in further regulatory interactions. The system of communications between RNA molecules constitutes an information subsystem of the cell.

The interactions between RNA molecules are reproduced using memories of such communications. These memories are the DNA molecules organized into the genome of the cell. The DNA molecules produce RNA molecules by interacting with proteins and RNA molecules. The produced RNA molecules participate in RNA interactions and ultimately lead to the generation of various regulatory and protein encoding RNA molecules and finally to proteins within the cell. The DNA molecules constitute a secondary memory subsystem of cell consisting of long-term memories (the DNA molecules usually survive with minor changes during the whole lifetime of the cell). In bacteria and archaea the DNA contains mostly segments which encode RNA molecules which lead to protein generation. In higher organisms the DNA contains a large amount of non-protein coding segments, which are believed to participate in regulatory interactions between DNA segments [40]. These regulatory interactions lead to the production of appropriate RNA molecules in appropriate quantities. The regulatory interactions between DNA segments constitute a communication system, which can be seen as the secondary information system of the cell (i.e., the information subsystem regulating RNA interactions).

Following the abstract communication systems theory interpretation we identified two levels of memory subsystems within the cell constituted by RNA and DNA molecules, providing memories of protein interactions (short-term memories) and RNA interactions (long-term memories), respectively. These memories generate and information subsystem made of interactions between RNA molecules and segments of DNA molecules, which regulate the expression of memories under their control including the creation of new memories (e.g., spliced RNA molecules, simultaneous production of various combinations of RNA molecules). The long-term memories contain a blueprint description of the cell, constituting the self-model of the cell.

The cell performs self-monitoring through the interactions between proteins, RNA molecules and segments of the DNA [46]. These communications trigger further communications which reference them. Some of the protein interaction

communications lead to interactions with RNA molecules. The results of such interactions can be seen as short term memories (i.e., possibly modified RNA molecules). These short term memories participate in RNA interaction communications contributing to the selection of appropriate RNA molecules for translation into proteins. RNA interactions may generate RNA molecules or proteins that interact with DNA molecules (e.g., siRNA molecules), driving the DNA expression into RNA accordingly to the cell's needs. Other RNA molecules may even create new DNA molecules (e.g., RNA retroviruses), creating new memories of RNA interactions. The self-monitoring ultimately refers to the self-model (i.e., the DNA blueprint) in order to check the identity of intracellular communications (i.e., whether they are correct or not, or expected or less expected).

The actions of the cell system are sequences or spatio-temporal patterns of protein interactions (e.g., secretion of molecules into the extra-cellular space, movement of the cell). These actions aim to validate the correctness of earlier cell communications and implicitly to reproduce and expand the cell. The actions are generated using the memories and the information processing subsystems of the cell, which regulate the generation of the right proteins in the right amounts and right places, such that the appropriate cell actions can be generated.

The cell perceptions are the differences between the expected distribution of protein interactions and the actual distribution of these interactions. The first is specified through the blueprint of the cell, the second emerges from self-monitoring identity check communications. For example the presence or absence of antibiotics in case of bacteria may lead to very different patterns and distributions of protein interactions (i.e., in the presence of ribosome-blocking antibiotics many wrong proteins are produced, generating many protein interactions, which do not fit into the system of protein interactions of the cell). The different pattern and distribution of protein interactions leads to changed patterns of RNA-protein expression and possibly to change patterns of DNA-RNA expressions (e.g., previously inactive DNA segments are expressed in neurons after long-term potentiation [12]).

Faults, errors and failures occur in cells. Faults are interactions between proteins that do not follow the language of the cell (i.e., the set of regular interactions). For example such interactions may lead to protein malformations like prions, which are proteins having a wrong conformation, preventing them from their regular interactions. Faults may lead nowhere, and in many cases the cell can simply ignore them, eliminating the possibly hazardous results of faulty interactions (e.g., toxins produced by such wrong interactions). Errors occur, when interactions happen according to the rules, but they cannot be continued by further appropriate interactions. For example, in the case of bacteria in the presence of antibiotics many appropriate protein interactions cannot be followed by such interactions because of the lack of appropriate proteins. When errors happen at the large scale, and large part of the cell system halt the cells may experience failure, which may lead to the disintegration of the cell (e.g., bacteria in presence of antibiotics).

Self-monitoring makes possible for the cells to perceive themselves and implicitly their environment and to select appropriate actions in order to increase their chance of self reproduction and expansion. In case of faults, errors and failures these are perceived in terms of deviations from the expected interactions and the cell invokes new parts of its memory subsystem using its information subsystem to generate responses to avoid catastrophic effects (i.e., the disintegration or large scale shrinking of the system) of these problems. Adaptive responses are generated by the cell reflecting the environment of the cell and its own state. Some of these adaptive responses lead to new memories and fuel the evolution of the system. For example, bacteria may develop antibiotic resistance in the presence of antibiotics, and may lose antibiotic resistance after residing in antibiotic-free environment.

Our analysis shows that cell systems constitute self-aware systems. They are able to monitor themselves and produce adaptive responses reflecting the state of their environment and of themselves. They are also able to build upon these adaptive responses and innovate themselves evolving into new forms adapted to their environment in such ways that increases their chance to reproduce and expand.

3.2 Organizations and Bureaucracies

Human communications (verbal, written, gesture based, etc.) constitute a communication system, which is the human society. Human society has many subsystems, like subsystems defined by natural languages (e.g., English, Chinese, etc. societies), and subsystems defined by their specific logic, with associated functionality in the context of their society (e.g., political system, legal system, economic system, etc.) [16], [16], [41], [53]. Organizations constitute subsystems within many parts of the human society, e.g., political parties within the political system, companies within the economic system, and universities in the higher education system.

Organizations can be viewed as communication systems made of communications between humans, these humans acting as communication units for the organization [4], [10], [15], [44]. The organization is defined by its own language, which restricts the distributions over possible continuation communications, conditioned by previous communications. The language of the organization is usually described in terms of statutes, regulations, contracts, rules (rules differ from regulations in the sense that rules have a limited range of applicability and restrict the relationship between a few factors, while regulations are large consistent sets of rules with a wide range of applicability and constraining the relationships of many factors), and rituals (interpreted in a wide sense, including all kinds of expected behavioural patterns). The language of the organization imposes structures (i.e., restrictions on communications) within the organization (e.g., organizational hierarchies).

Organizations produce memories that contribute to the success of reproduction and expansion of the organization. Such memories are products and services, which represent in a compressed form the human communications that lead to their production, written paper and electronic records of organizational com-

munications, and repeatedly told stories of organizational events. The memories of the organization are organized into a memory subsystem by communications about these memories in form of generating organizational rituals, traditions, rules and regulations. The latter constitute the foundation of the identity checking information subsystem of the organization. Products, services and records of organizational communications constitute short-term memories of the organization, they being referenced for relatively short time period. Memories about processing of short-term memories, like rituals or regulations are long-term memories of the organization, they persist for long time periods and are referenced during this time when memory communications are processed. The long-term memories of the organization describe a self-model of the organization (e.g., the constitution of a party describes the organizational units of the party, the rights and responsibilities of these units and of their members, the decisional procedures, etc.).

Large organizations have well developed information subsystems, which use and generate memories and check the identity of organizational communications by referencing traditions, rules and regulations. The information subsystem of such organizations takes the form of organizational bureaucracy. Thus, the bureaucratic subsystem of the organization deals primarily with production of records, assessment of records, and production of new identity checking communications in forms of particularized orders based on existing regulations, formulation of new rules, contracts and regulations. The organizational bureaucracy in particular generates the long-term memories of the organization.

Organizations self-monitor themselves by checking the identity of organizational communications and by generating records of organizational communications for later identity checks. Organizational bureaucracies perform to large extent the self-monitoring of the organization by generation and assessment of records of organizational communications. Self-monitoring drives the appropriate application of rules by referencing long-term memories of the organization (the organization's blueprint) contained in regulations defining the what is allowed and what is not in the context of organizational communications.

The actions of an organization are organizational communications that modify the communications in the environment of the organization (e.g., selling products, conquering a part of a market, changing a part of the political discourse etc.). Communications constituting organizational actions follow the rules of the organization (e.g., rules describing the delivery of a service). The perceptions of the organization are the differences between the expected distribution of organizational communications and their actual distribution. These perceptions are assessed using memories of earlier communications usually by the organizational bureaucracy. Actions generate perceptions and perceptions generate actions all aiming to increase the reproduction and expansion ability of the organization. If the rules of the organization (i.e., the identity of the organization) match its environment the perceptions and actions will allow the organizations to recruit more organizational communications, which usually materializes in the expansion of the organization in the sense of having more humans communicating

within the organization, more products and services sold and gaining new parts of markets.

Organizations also experience faults, errors and failures. Faults are those organizational communications which do not follow the rules of the organization. These many times are eliminated without providing reference for many further organizational communications. Errors occur when organizational communications follow the rules, but they cannot be continued due to environmental constraints. For example, an inappropriate marketing campaign does not lead to increase in sales. A major functional role of the organizational bureaucracy is to discover faults and errors and limit their effects on the organization. By applying identity checks (i.e., assessing memories of communications - records to find out whether they comply with the rules) and forming expectations about future communications according to the rules an effective organizational bureaucracy can spot faulty communications and recognize errors. In case of errors the bureaucracy aims to find the roots of it by analyzing records of organizational memories, i.e., those communications that led to the generation of the error, and generates new identity checks, i.e., new communications about rules that are intended to be used to prevent the occurrence of similar errors. Failures happen when errors halt the continuation of communications within a large part of the organizations. Failures may lead to the disintegration of the organization, and usually lead to major restructuring of it, changing rules and regulations, and reorganizing the organizational bureaucracy.

Organizations adapt to their environment in response to faults, errors and failures by changing their rules, regulations, structure, and possibly even their identity (e.g., companies may move completely from one market to another). These changes in organizations often take the form of adding and modifying the regulations of the organization, and many times this leads to standardisation / simplification of organizational communications. Such changes trigger the reorganization of the organization, provision of new products and services, the change of its bureaucracy and possibly of its identity. The changes may be beneficial or not for the organization. In a competitive environment, where many organizations compete for communications generated by the same communication units (i.e., humans) maladaptive change leads to the shrinking and possibly to the dissolution of the organization. In less competitive environment (e.g., state monopolies or very large corporations) maladaptive changes are likely to lead to overgrowing bureaucracy as new regulations are put in place to prevent earlier errors and failures triggering the growth of the organizational bureaucracy in charge of generating and imposing regulations and checking adherence to regulations.

Organizations are self-aware systems, which monitor themselves using their memories and information subsystem (i.e., the organizational bureaucracy), sense their environment and act upon their environment, and adapt to their environment. Organizational adaptations are generated by the organizational bureaucracy in form of new or revised rules and regulations aimed to prevent faulty communications, errors and the occurrence of failures, guaranteeing increased ability of the organization to reproduce and expand.

4 Self-Aware Software

Computer software drives the computer hardware and it is present in increasingly many machines and appliances, including cars, fridges, and mobile phones. Computer software is made of computer programs written in some programming language (e.g., Java, C++) and executed on some computer hardware, which is able to understand the programs and translate them into behaviours of the machine. In the early times (1950s - 1960s) computer programs were independent of each other, were executed sequentially, and mainly dealt with processing and transforming some data records and possibly producing some additional data records. More recently computer software is composed of many concurrently active components which communicate with each other.

The usual way is to view computer software in functional terms, associating with them a set of functions that they can perform and which might be useful for their user (e.g., word processing, graphics generation, web browsing, etc.). An alternative way to consider computer software is to view them as part of the human society interpreted as an abstract communication system. In this context computer programs can be seen as recorded memory of human communication (i.e., the communication of the programmers with computers, which transformed their typed communications into stored records containing the program). This memory can be recalled by using a computer (or some computing hardware in general), and the recalled communication will be used to process new society communications (human or indirectly related to human communications) and generate possibly new communications and records of earlier communications. The software is part of the information subsystem of the society allowing the processing of memories (i.e., electronic records of earlier communications) and to generate new society communications (e.g., a printed page, a spoken sentence, or stored data). Unsurprisingly, computer software is used to a very large extent in the context of information subsystems (i.e., bureaucracies) of various social subsystems, like companies, government agencies, and other kinds of organizations. (We note that computers themselves are products of human communications and can be seen as memories of these communications that led to the design and assembly of them.)

In alignment with the above presented view, computer software can be seen as a communication system of many processes or components executed on computer hardware. In the context of object-oriented software, we may consider each object (i.e., an instantiation of a class, which is specified in the code of the software) as a communication unit, and view the software itself as the set of interactions between these communication units. In current large scale software systems such objects appear and disappear frequently, communicating with many other objects, some of these being created by other programs. Some objects may create copies of themselves or other objects on distant hardware, and may reproduce and expand the communication system of which they are part of (e.g., spyware).

Here we aim to investigate the ways by which computer software may become self-aware. First let us review the features of self-aware communication

systems that we highlighted in the previous section of the paper. Systems with self-awareness all have short- and long-term memories and identity check communications that reference these memories. Memories provide the foundation for self-monitoring, which is performed by the information subsystem of the system using identity check communications and generation of new long-term memory communications by processing existing memory communications. The system of long-term memories defines a self-model of the system, which is ultimately referenced by identity check communications. Self-aware systems perceive their environments in terms of differences between expected and actual system behaviour, perform actions by which they modify their environment, and aim by their perceptions and actions to reproduce and expand the system. In principle these systems may expand infinitely, although in practice they may reach some limits of their expansion. Self-aware systems adapt to their environment by changing themselves in response to experienced faulty communications, errors and system failures. System adaptations change the information subsystem of the system, by imposing new or modified identity checks, or smaller or larger scale reorganizations of the information subsystem, changing the rules defining the identity of the system. The rest of the section will analyse each of the listed features of self-aware systems in the context of software systems.

In the case of software systems we can identify short and long term memories in forms of data and software code. The data is allowed to change frequently in most cases. Stored data represents to some extent the memories of communications between objects (although not all communications are represented by stored data). The software code constitutes long term memories (the blueprint or self-model of the system), which allow the recreation of objects many times, and store the rules of communications that lead to creation of these objects in computer memories. The software code is usually not changing. More recently new software systems emerged, which allow to some extent the change of their code by incorporating new pieces of code into the system's long term memory, e.g., security patches installed by human user or possibly installed automatically. These changes in the long term memory of the system represent adaptations based on processing memories of recent communications within the system, which led to faults, errors and failures. We need to note that all these changes to the software code reference human communications, which created the software patches. Recent software environments like Smalltalk, Java and .NET implement computational reflection [21], [42], which allows in principle changes to the program code, but the range of practically allowed changes is very limited (e.g., asking for names and types of exposed variables and methods in newly added components [24]). Elementary structural changes, like using new methods available in newly added components are allowed in the context of component-based programming [34]. Software systems usually do not create memories of communications between objects (except in cases of faults or errors that can be interpreted by exception handling components of these objects) and in consequence are unable to develop an information subsystem that would process memories and possibly generate new memories, including changes to the long term memory of the

system. However, we note that recent advances in the area of aspect-oriented programming [19], [33] allow the software system to track communications between objects. This can be seen as a critical first step towards the creation of memories of communications between objects (see logging of object communications in the context of aspect-oriented programming). In order to make software systems able to develop self-awareness it is very important to expand their memory subsystem to provide the foundation for an information subsystem within the software system.

Self-monitoring is performed by the information subsystem of self-aware systems. In case of software systems we can find some basic level of self-monitoring in form of correctness monitoring using methods of exception handling, type checking and execution pattern matching (in context of aspect-oriented programming). These can be seen analogous to basic identity check communications in self-aware systems. These communications usually check the correctness of data; they do not check the correctness of communications between objects or processes in any more general sense (this is usually done during the compiling of the software by checking that the long term memory of the software system conforms the rules of the programming language in which it is written). Recent advances in aspect-oriented programming [19], [33] and design patterns [28] indicate significant progress in terms of self-monitoring, by allowing tracking of object communications (aspect-oriented programming) and using blueprints of relatively simple communication patterns to enforce appropriate communications (design patterns; we note also the similar role of 'interface'-s in the context of Java). Self-aware software systems need much larger scale self-monitoring comparable to self-monitoring of natural self-aware systems. Self-monitoring should be based on memories of communications that happen between objects and processes, should check the identity of these communications referencing identity check communications of the software system (in particular the self-model of the system, the software code), and generating new memory and identity checking communications (including new parts of the software code).

In case of natural self-aware systems the identity check communications generate a large part of communications within the system, and we may see the system emerging from such identity check communications. Due to the Popper principle the system's correctness cannot be proven, and identity checking (or correctness proving) communications may continue infinitely, guaranteeing the reproduction and expansion of the system. Existing software systems have basic mechanisms to check the correctness of data (e.g., type checks, exception handling methods), but most of them do not check the correctness (or identity) of communications between communication units (i.e., objects) during runtime and consequently do not lead to the expansion of the software system interpreted as a communication system. We note that run-time type checking is present in Smalltalk and also to some extent in Java, and the identity check of communications may become a practical possibility following current trends in aspect-oriented programming [19], [33] and use and development of design patterns [56]. In case of software systems reproduction and expansion can be seen

to some extent in form of re-use (template libraries [29], design patterns [26], component-based programming [30]) and in form of installation and running of many copies of them. Some of them, like software viruses, are able to replicate themselves by creating new copies of their own code, which is run on the same or on different computer hardware. Importantly, software systems usually do not adapt autonomously to their environment (software patches distributed automatically over the Internet can be seen as a basic form of adaptive change triggered by human communications generating the patch) or their adaptation is very limited (recognizing methods and types of variables of a new component). Perhaps the most autonomous adaptive behaviour can be seen in the context of implementation of continuation in programs developed using functional languages, which allows capturing of continuations of executions and the adaptive recombination of such captured continuations [20], [7]. To overcome the present limitations of software systems, and to develop self-aware software systems a major change is needed in the design and analysis of software systems. While current systems are designed and analysed in terms of their functionality imposed on them in the context of their environment, self-aware software systems should be designed and analysed using a 'from within' view, which permits the development of the system such that its own identity check communications lead to the emergence of the system. Self-aware software systems should aim to reproduce and expand themselves as communication systems and they should perform their intended and externally imposed functionality by adapting to their environment and reproducing and expanding within this environment. Steps that may lead to such systems include works on cellular automata [25], multithread parallel systems [59], software viruses [35], the concept of continuation [20], [7], development of persistent systems [8], self-monitoring and self-healing systems [6], component based programming [30] and aspect-oriented programming [19], [33], [54].

Actions of software systems are sequences of communications with effect on the environment of the system. These effects can be triggered actions of human users (e.g., pressing a button or clicking the mouse on a certain place of the screen) or communications originating from other software systems (e.g., arrival of data through the Internet connection). The actions of the software system follow the rules established by the self-model of the system (i.e., procedures described in the software code), but the referencing of the self-model happens mostly during the compilation of the code into objects (notable exceptions are programs written in scripting languages, e.g., the use of the function 'eval' to include new code from an additional file). In contrast, natural self-aware systems reference their self-model communications frequently during the identity check communications. They also generate short-term memories of their action communications, which are checked for their identity by referencing other memory communications, self-model communications, and by generating new memory communications. Perceptions of software systems are the detections of changes in their environment. Typically such perceptions are implemented in the form of 'if-then-else' rules or multiple variants of this (e.g., 'switch', 'case-of'), or

in form of exception handling (i.e., equivalent of a final or possibly branching 'else'). The expectation about possible communications is usually characterized by a flat prior distribution, i.e., there is no prior information represented in the encoding of the perception interpretation code. The difference between the flat prior expectation and the experienced distribution (i.e., one of the possibilities appears with certainty, while others have zero experienced probability) triggers communications within the software systems. Similarly to the case of actions, perception communications reference the self-model of the software system only at time of compilation of the objects. Natural self-aware systems have expectations based on prior experience. These expectations are usually characterized by non-flat distributions over the space of possibilities. Prior experience changes the expectations adaptively, so that the system can generate the most appropriate communications in response to its perceptions. Self-aware systems also reference their self-model and memory communications frequently when they generate perception triggered communications. To turn software systems into self-aware systems they need to have frequent references to their short- and long-term memories, adaptive expectations encoding non-flat prior distributions of possibilities, and more flexibility of the behaviour of the software system that might be achieved by on-line adaptation of the self-model (i.e., the software code) and frequent recompilation (or partial recompilation) of objects.

Works on dependable systems and software reliability [9], [38] created an elaborated theory of faults, errors and failures in the context of software systems. The most common mechanisms to deal with faults are the data validation combined with roll-back and the exception handling methods. The roll-back is triggered by invalid data and restores the previous correct state of the system. In case of databases the roll-back restores previous data that satisfied the validity checks before, in case of more complex software systems recovery blocks executed in parallel may be used to restore a valid state of the system [38]. Exception handling methods include the handling of invalid data, type mismatch, and invalid access rights, providing essentially an 'if-then-else' type solution of dealing with unexpected communications grouped in one or a few categories. Related recent work addresses the concept of 're-start', i.e., when does a program or communication need to be re-initiated [65]. 'Re-start' can handle in principle a general class of errors, when the expected continuation communications do not happen. Natural self-aware systems check the identity (validity) of communications permanently by referencing their short- and long-term memory communications and creating new identity check communications. Identity check communications eliminate most of the faulty communications, errors are limited by structures that may change adaptively, and failures trigger major adaptive changes in the self-model of the system. This suggests that self-aware software systems need to expand their response repertoire to deal with faults, errors and failures, by employing frequent identity check communications to validate data and communications between objects, including communications referencing long-term memory communications (i.e., the software code), use of adaptive structures, re-start methods and parallel multi-thread execution to limit errors

and to create slack resources (i.e., enough many communications so the system can continue its own reproduction even if many communications lead to faults and errors), and by adapting its own self-model in response to errors and failures by changing and adding to the software code.

Adaptation is a key feature of natural self-aware systems. Adaptation includes the generation of appropriate actions and forming of appropriate perceptions in order to increase the reproduction and expansion ability of the system, and also the modification of the self-model of the system in response to errors and failures experienced by the system. Software systems are able to form appropriate perceptions and generate appropriate actions to some extent, but they are unable to adapt their self-model by themselves during their existence in response to errors and failures experienced by them. Software systems are most adaptive in terms of stored data that is changing frequently and to lesser extent in terms of changing access interfaces of objects (e.g., using methods of a newly added component). Adding new security patches and updates is a relatively new way of adaptation of software systems, which changes the self-model of the software system in response to errors and failures. At the same time we have to note that all these adaptations are triggered by communications with humans (even in the case of automated updates the new modules are written by human software developers). Just-in-time compilation, run-time type checking, and garbage collection are the beginnings of more adaptive behaviour, which allow adaptive changes with respect to the actively referenced part of the self-model and introduce more frequent references to the self-model. The concept of 're-start' [65] represent a new pathway that may lead to novel ways of adaptation in software systems in response to errors (i.e., lack of continuation of communications as expected). Analysing natural self-aware systems suggests that a much wider range of adaptive responses is needed in software-systems to achieve self-awareness. Memories of runtime interactions between objects/processes need to be created. These should generate new communications by referencing identity check communications and other memory communications in order to check the identity (validity) of these interactions, eliminating faulty communications when they appear. Identity check communications should reference frequently the self-model of the system and should lead to changes in the self-model of the software system (i.e., the software code) in response to errors and failures. These changes of the self-model should manifest in new structural constraints imposed on communications between objects, possibly in simplification / standardisation of object communications, in specialisation of communications between objects, leading to the emergence of specialist subsystems of the software system, and possibly in the generation or recruitment of new types of objects into the software system.

In general, comparison of software systems with natural self-aware systems suggests that current software systems need many changes to become similar to self-aware systems and possibly to become self-aware software systems. A fundamental difference from current practice that seems to be needed is that self-aware software systems need to be designed using a 'from within' perspective, aiming to build a system that reproduces and expands itself and becomes

associated with externally perceived functions by adaptation to its environment. The self-reproduction and expansion of the system should happen by generating short-term memories of object communications and endless number of identity checking communications that reference short- and long-term memory communications. The self-aware software system should be able to adapt to its environment by choosing appropriate actions and perceptions, and also by adapting its own self-model contained in its long-term memories the software code. The adaptation of the self-model is critical for self-awareness, and should include changes to the code, generation of new structures, and incorporation of new classes and generation of new objects.

5 Will Self-Aware Software Become a Reality?

In this section we discuss to what extent we expect that self-aware software that satisfies our descriptions outlined in the previous section will become an existing reality. We discuss aspects that we consider critical in terms of resources and approaches.

In our view self-aware software systems will aim in principle to grow without any limit. In practice the environmental constraints may limit this growth very much as they do this in the case of existing natural self-aware systems (e.g., bacteria, animals, organizations). In most cases of natural systems the growth limits of the system imposed by scarcity of resources needed for communication units and communications do not limit the growth of the system in practical sense, and the habitable environment of the system is apparently infinite from the system's point of view. We believe that in order to generate self-aware software we need to achieve the state in which the habitable environment of the software system becomes apparently infinite. The habitable environment of software systems is provided by available hardware and software components (i.e., objects, processes run on the large amount of hardware). At the moment this habitable environment is relatively small, which does not offer apparently limitless resources for the expansion of software systems. Simple replicating software, like viruses spread on the Internet, are able to populate very rapidly the existing available software/hardware environment, exhausting their resources for growth, and collapsing as a system. In order to have the environmental conditions for self-aware software we need many magnitudes increase of available software and hardware resources, so that the environment provided by them allows practically limitless growth for self-aware software systems.

As we noted earlier it is very critical that self-aware software is developed using a 'from within' approach. A similar approach can be seen in the case of the Smalltalk [27] programming language, in which everything is an object, including the development environment, and programs are built by combining existing objects and building new objects [22], [61]. The main difference of the proposed approach from most current software development approaches is that it does not subordinate the development of the software system to the function imposed on the system from outside. The proposed 'from within' approach will

allow to associate functions to the software system as it reproduces and expands under environmental constraints. This does not mean that software components (e.g., objects) or the software system is not developed to fulfil some specific role; it rather means that components, communications, and the self-model of these is selected and composed such that the system emerges to fulfil its designated role. We do not know at the moment what would be the detailed features of the software development 'from within', but we think that current trends in component-based programming and aspect-oriented programming are pointing towards the 'from within' approach to software development.

Self-aware software in our view will be the sum of communications between objects that follow the rules described in the self-model of the software system (i.e., the software code). We believe that such systems will be based on re-use of many existing components (runtime objects, coded classes), and will recruit communications involving such existing components. Similar to human organizations or cells the focus of the system will be only partly to produce its communication units (humans and proteins, respectively), but to a larger extent will be to use the existing ones, involving them in communications according to the rules of the system. This means that self-aware software systems will become a possibility when an apparently infinite number (obviously objectively this will be still a finite number) of available software components will exist in an apparently infinite supporting environment provided by computer hardware. The self-aware software system may still produce many objects according to its own self-model so it can build up itself, but it will be necessary for the software system to be able to incorporate in itself communications with existing objects. Existing objects of which code is not part of the software system may provide new segments for the adaptive self-model of the software system, making it better able to reproduce and expand in its environment. Today we already can see beginnings of a hardware/software environment which might become able to support the development of self-aware software according to our vision. The increasing re-use of software components, the availability of template libraries [29], design patterns [26], [56], and in some cases of their source code (i.e., open source versions of them or the bytecode of Java objects), the development of standardised communication interfaces and communication protocols between objects, hint that it might not be in the too distant future that building software by combining existing objects and source code will be possible. An early step towards self-aware software might be the building of software systems by specifying the design of communication patterns between existing components, without adding any new component or explicitly rewriting the behaviour of existing components (see design patterns).

As we pointed out in the previous section natural self-aware systems extensively use memory communications, and they create memories of many communications. Having these memory communications provides the foundation for their self-monitoring by their information subsystem and for the adaptation of their self-model. The critical missing component of software systems in this respect is the lack of creating memories of communications between objects/processes,

however recent advances in aspect-oriented programming address this issue at least at a basic level by allowing the logging of object communications [19]. Consequently, the analysis of natural self-aware systems suggests that in order to develop self-aware software systems we need to have memories of communications between objects and these memory communications need to be referenced frequently by communications constituting the software system. Although it is not completely clear for us how the creation of object communication memories should be implemented, we believe that such memory communications can be realized in the form of creating new objects which generate repeatedly a representation of the memorized communication (or combining simple existing objects by creating re-occurring communications between them or by extension logging communications implemented in the context of aspect-oriented programming). These memories should work as short-term memories, implying that they can be discarded after a relatively short time, making their sustaining components (i.e., objects which produce the memory communications) available to store new memory communications or to participate in other communications. The development of these short-term memories will bring us closer to the method of 'from within' for software system development.

Natural self-aware systems grow to a large extent due to identity check communications. In other words, due to the Popper principle, the asymmetry of true and false decisions about the identity/validity of communications, systems that are able to exist check their identity for infinity (at least in principle), systems that cease to exist conclude at some moment that their identity does not exist by ceasing the continuation of their communications. In our opinion self-aware software systems should expand largely due to identity checking communications, which check the identity/validity of communications between objects, generating new communications between objects, including memory communications, and guaranteeing the reproduction and expansion of the system according to its own rules. We believe that the 'from within' method of development of self-aware software will include methodology that allows the expansion of the software system by the generation of identity check communications. As we already noted, the function of the self-aware software system will be achieved by adaptation to its environmental constraints, which limit its growth and trigger the generation of additional identity check communications by contributing to the generation of some actual communications that differ to some extent from the expected communications (at least in their distribution). We note that our view of growing the software system by identity check communications is similar to the method of development of programs by elaborating the proof of initial statements representing the problem and assumptions [32], [43], [60]. The main difference between our approach and program development by proof is that in our view the proving process should continue infinitely in case of self-aware software systems, while in the context of program development by proof the proving process ends by proving the correctness of the program. While program development by proof operates in a world supposed to be static and completely known (i.e., represented by the assumptions and the problem statement, of which validity is assumed to

be provided), in our view self-aware software develops in a world that is infinitely complex and variable, the software system being able to capture (i.e., describe in a complementary sense) at any time a limited part of this world. In the context of our assumption, if the proving process halts that means the end of the software system, and the infinite continuation of the proving process means the reproduction and possibly expansion of the software system.

Adaptation of the self-model of natural self-aware systems is their core critical feature. Existing software systems are limited in adapting their self-model as we discussed in the previous section. If the previously discussed issues will be solved according to our expectations, the adaptation of the software code in response to faults, errors and failures will become possible. The use of existing objects and the access to their source code (their self-model) will allow the inclusion of new parts into the self-model of the software system. Self-monitoring, memories of runtime interactions, and frequent references to the self-model of the system, and accessibility of new self-model components will make possible to search for roots of errors, to find appropriate simplifications and standardisations, and to find appropriate changes to the self-model of the system, which will increase its reproduction and expansion ability.

Finally, an interesting question is whether the self-aware software will show anything similar to human consciousness, including feelings and emotions. In our view there is a wide variety of natural self-aware systems, which include bacteria, plants, simple and complex animals, social animals like humans, human organizations, and possibly other natural self-aware systems. These systems perform their self-awareness in many different ways, but all maintain the critical features of self awareness (including the adaptive change of their own self-model). One of these systems, the humans, show consciousness, while others may show similar features to some extent (e.g., animals), but others do not show any behaviour that would resemble human consciousness (at least in terms of communications/interactions with humans). This indicates that self-aware software systems may not show any behaviour that would resemble human consciousness, and in principle there is no reason why we should expect any such behaviour. Our view is that self-aware software systems (if they become reality) will show the critical features of self-awareness, but they will not behave like conscious humans.

References

1. Agrawal, N, et al. (2003). RNA Interference: Biology, Mechanism, and Applications. *Microbiology and Molecular Biology Reviews*, 67: 657-685.
2. Alonso, E, Kudenko, D and Kazakov, D (eds.) (2003) *Adaptive Agents and Multi-Agent Systems*. Springer-Verlag, Berlin,
3. Andras, P and Andras, CD (in press). Protein interaction world - an alternative hypothesis about the origins of life. To appear in *Medical Hypotheses*.
4. Andras, P and Charlton, BG (2004). Management from the Perspective of Systems Theory. In *Proceedings of Practising Philosophy of Management - 2004*.

5. Andras, P and Charlton, BG (2002). Democratic Deficit and Communication Inflation in the Health Care System. *Journal of Evaluation in Clinical Practice*, 8: 291-298.
6. Appavoo et al. (2003). Enabling autonomic behavior in systems software with hot swapping. *IBM Systems Journal*, 42: 60-76.
7. Ariola, ZM, Herbelin, H, and Sabry, A (2004). A type-theoretic foundation of continuations and prompts. *ACM SIGPLAN Notices*, 39: 40-53.
8. Atkinson, MP and Morrison, R (1995). Orthogonally persistent object systems. *VLDB Journal*, 4: 319-401.
9. Aviziensis, A, Laprie, JC, and Randell, B (2001). Fundamental concepts of dependability. Newcastle University Report no. CS-TR-739.
10. Barnard, CI (1938). *The Functions of the Executive*. Harvard University Press, Cambridge, MA.
11. Barr, A and Feigenbaum, EA (1989). *The Handbook of Artificial Intelligence*. Volume II, 2nd printing, Addison-Wesley, Reading, MA.
12. Behnisch T, Matsushita S, and Knopfel T (2004). Imaging of gene expression during long-term potentiation. *Neuroreport*, 15: 2039-2043.
13. Born, R (Ed.) (1989). *Artificial Intelligence: The Case Against*. Routledge, London, UK.
14. Cazzola, W, Stroud, RJ, and Tisato, F (2000). *Reflection and Software Engineering*. Springer-Verlag, Heidelberg.
15. Charlton, BG and Andras, P (2004). The Nature and Function of Management - a perspective from systems theory. *Philosophy of Management*, 3: 3-16.
16. Charlton, BG and Andras, P (2003). *The Modernization Imperative*, Imprint Academic, Exeter, UK.
17. Cohen, PR and Feigenbaum, EA (1989). *The Handbook of Artificial Intelligence*. Volume III, 2nd printing, Addison-Wesley, Reading, MA.
18. DiStefano, A, Fragetta, M, and Tramontana, E (2003). Computational reflection for embedded Java systems. LNCS 2889, Springer Verlag, Heidelberg, pp.437-450.
19. Douence, D, Fradet, P, and Sudholt, M (2004). Composition, reuse, and interaction analysis of stateful aspects. In: *Proceedings of AOSD 2004*, ACM, pp.141-150.
20. Duba, B, Harper, R, and MacQueen, D (1991). Typing first-class continuations in ML. In: *Proceedings of the 18th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, ACM, pp.163-173.
21. Ferber, J (1989). Computational reflection in class based object-oriented languages. *ACM SIGPLAN Notices*, 24: 317-326.
22. Foote, B and Johnson, RE (1989). Reflective facilities in Smalltalk-80. *ACM SIGPLAN Notices*, 24: 327-335.
23. Frank, MR and Szekeley, P (1997). Adaptive forms: an interaction paradigm for entering structured data. In: *Proceedings of the 3rd International Conference on Intelligent User Interfaces*, ACM, pp.153-160.
24. Furicht, R, Prahofor, H, Hofinger, T, and Altmann, J (2002). Components: A component-based application framework for manufacturing execution systems in C# and .NET. In: *Proceedings of TOOLS Pacific 2002*, ACM, pp.169-178.
25. Gacs, P (2001). Reliable cellular automata with self-organization. *Journal of Statistical Physics*, 103: 45-267.
26. Gamma, E, Helm, R, Johnson, R, and Vlissides, E (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA.
27. Goldberg, A and Robson, D (1983). *Smalltalk - 80. The Language and its Implementation*. Addison-Wesley, Reading, MA.

28. Grothoff, C (2003). Walkabout revisited: The runabout. In: LNCS 2743, pp. 103-125.
29. Jarzabek, S (1995). From reuse library experiences to application generation architectures. ACM SIGSOFT Software Engineering Notes, 20: 114-122.
30. Jarzabek, S and Knauber, P (1999). Synergy between component-based and generative approaches. ACM SIGSOFT Software Engineering Notes, 24: 429-445.
31. Jennings, NR, Sycara, K, and Woolridge, M (1998). A roadmap of agent research and development. *Autonomous Agents and Multi-Agent Systems*, 1: 7-38.
32. Jones, CB (1972). Formal development of correct algorithms: An example based on Earley's recogniser. In: *Proceedings of ACM Conference on Proving Assertions about Programs*, ACM, pp.150-169.
33. Katara, M and Katz, S (2003). Architectural views of aspects. In: *Proceedings of AOSD 2003*, ACM, pp.1-10.
34. Killijian, M-O, Ruiz, J-C, and Fabre, J-C (2002). Portable serialization of CORBA objects: a reflective approach. *ACM SIGPLAN Notices*, 37: 68-82.
35. Kinzle, DM and Elder, MC (2003). Internet WORMS: past, present, and future: Recent worms: a survey and trends. In: *Proceedings of the 2003 ACM workshop on Rapid Malcode*, ACM, pp.1-10.
36. Landauer, C and Bellman, KL (2001a). New architectures for constructed complex systems. *Applied Mathematics and Computation*, 120: 149-163.
37. Landauer C and Bellman, KL (2001b) Self-modelling systems. In: LNCS 2614, pp.238-256.
38. Lee, P.A. and Anderson, T. (1990). *Fault Tolerance. Principles and Practice*. Wien: Springer-Verlag, 2nd ed.
39. Lee, Y et al. (2003). The nuclear RNase III Droscha initiates microRNA processing. *Nature*, 425: 415-419.
40. Levine M and Tjian R (2003). Transcription regulation and animal diversity. *Nature*, 424: 147-151.
41. Luhmann, N (1996). *Social Systems*. Stanford University Press, Palo Alto, CA.
- Charlton, BG and Andras, P (2003). *The Modernization Imperative*, Imprint Academic, Exeter, UK.
42. Maes, P (1987) Concepts and experiments in computational reflection. *ACM SIGPLAN Notices*, 22: 147-155.
43. Maghrabi, T and Golshani, F (1992). Automatic program generation using sequent calculus. *Proceedings of the 1992 ACM Annual Conference on Communications*, ACM, pp.73-81.
44. March ,JG and Simon, HA (1993). *Organizations*. Blackwell, Cambridge, MA.
45. Maturana HR, and Varela, FJ (1980). *Autopoiesis and Cognition : the realization of the living*. D. Reidel Publishing Company, Boston.
46. Mattick, JS and Gagen, MJ (2001). The evolution of controlled multitasked gene networks: The role of introns and other noncoding RNAs in the development of complex organisms. *Molecular Biology and Evolution*, 18: 1611-1630.
47. Meister, G and Tuschl, T (2004). Mechanisms of gene silencing by double-stranded RNA. *Nature*, 431: 343-349.
48. Miller, JG (1978). *Living Systems*. McGraw-Hill.
49. Newell, A (1990). *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.
50. O Cinneide, M and Nixon, P (2001). Patterns and evolution structures: Automated software evolution towards design patterns. In: *Proceedings of the 4th International Workshop on Principles of Software Evolution*, ACM, pp.162-165.

51. Ortin, F and Cueva, JM (2003). Non-restrictive computational reflection. *Computer Standards & Interfaces*, 25: 241-251.
52. Ortin, F, Lopez, B, and Perez-Schofield, JBG (2004). Separating adaptable persistence attributes through computational reflection. *IEEE Software*, 21: 41-49.
53. Pokol, B. (1992) *The Theory of Professional Institution Systems*. Felsooktatasi Koordinacios Iroda, Budapest.
54. Popovici, A, Alonso, G, and Gross, T (2003). Spontaneous container services. In: LNCS 2743, pp. 29-53.
55. Raghavan, A et al., (2002). Genome-wide analysis of mRNA decay in resting and activated primary human T lymphocytes. *Nucleic Acids Research*, 30: 5529-5538.
56. Rising, L (1998). *The Patterns Handbook*. Cambridge University Press, Cambridge, UK.
57. Rosenblat F (1962) *Principles of Neurodynamics : Perceptrons and the Theory of Brain mechanisms*. Washington D.C., Spartan.
58. Simon, HA (1976). *Administrative Behaviour*. The Free Press, New York, NY.
59. Stunkel, CB, Sivaram, R, and Panda, DK (1997). Implementing multidestination worms in switch-based parallel systems: architectural alternatives and their impact. *ACM SIGARCH Computer Architecture News*, 25: 50-61.
60. Sun, Y-Q, Lu, R-Z, and Bi, H (1985). Program synthesis based on Boyer-Moore theorem proving techniques. *Proceedings of the ACM 13th Annual Conference on Computer Science*, ACM, pp.348-355.
61. Tanter, E, Noye, J, Caromel, D, and Cointe, P (2003). Partial behavioral reflection: spatial and temporal selection of reification. *ACM SIGPLAN Notices*, 38: 27-46.
62. Uhrmacher AM, Rohl M, and Kullick B (2002). The role of reflection in simulating and testing agents: An exploration based on the simulation system James. *Applied Artificial Intelligence*, 16: 795-811.
63. Ungar, D and Smith, RB (1991). SELF: The power of simplicity. *LISP and Symbolic Computation*, 4: 187-205.
64. Valetto, G and Kaiser, G (2002). A case study in software adaptation. In: *Proceedings of WOSS'02*, ACM, pp.73-78.
65. Van Moorsel, A and Wolter, K (2004). Analysis and Algorithms for Restart. In: *Proceedings of the Quantitative Evaluation of Systems, QEST 2004*, pp. 195-204., IEEE Computer Society.