Fundamental Insights into Holonic Systems Design

Paul Valckenaers and Hendrik Van Brussel

K.U.Leuven, Department of Mechanical Engineering,
Celestijnenlaan 300 B, B-3001 Leuven, Belgium
Paul.Valckenaers@mech.kuleuven.be

Abstract. This paper goes back to the origins of the Holonic Systems concept, a wording coined by Arthur Köstler [1] but actually based on fundamental insights from Nobel Prize winner Herbert Simon [2]. Simon's theme is limited rationality and its implications for the ability to create and sustain sophisticated artifacts in the dynamic and demanding environments that are characteristic for today's society. Holonic and multi-agent systems are amongst the most complex artifacts emanating from deliberate human design and development activities. Therefore, this paper presents these fundamental insights from Simon, augmented with more recent research results on complex adaptive systems [3], and discusses implications for the design of Holonic Multi-Agent Systems. In particular, the development of subsystems (holons) suited for incorporation into larger systems, at some later stage and without knowing these larger systems in much detail, is at the center of the discussions in this paper**.**

1 Introduction

Holonic and multi-agent systems are amongst the most complex artifacts created through deliberate human design and development activities. However, the human designers and developers, involved in these activities, often perceive their results as missed opportunities. This is especially true for the experts that thoroughly master and understand the technologies and subsystems within these complicated and sophisticated systems from the basic elementary building blocks up to the overall system itself.

This paper presents and discusses insights that reveal how this perception of missed opportunities is partly an illusion, which is mainly the contribution from Simon. Conversely, this paper shows how enhanced design principles and development guidelines, based on these insights, open the perspective of creating and maintaining superior systems.

The paper first presents the fundamental insights, revealed by research results on limited rationality [2] and on complex adaptive systems [3]. Next, design principles derived from these insights are presented and illustrated. Finally, some conclusions are given.

2 Limited Rationality and Holonic Systems

Simon's *Watchmaker's Parable* in [2] demonstrates how, in dynamic and demanding environments, the chances of emerging and surviving for systems composed of suit-

V. Mařík, R.W. Brennan, M. Pěchouček (Eds.): HoloMAS 2005, LNAI 3593, pp. 11 – 22, 2005.

able subsystems are vastly superior to systems composed from basic elements without stable intermediate states or subsystems. In Köstler's words, holonic systems are more likely to emerge and survive in dynamic environments than systems that would ultimately be superior but take too much design effort and development time. The latter systems simply are too expensive and, most importantly, obsolete long before they become operational.

Limited rationality – finite brainpower, bounded information processing and communication capacity – puts a ceiling on the speed at which elements can be integrated to build a system. When these elements are small, more time and effort is required to build a system of a given size. In combination with a dynamic environment, the resulting system is completed too late to be effective or competitive. Systems build from larger building blocks are superior in environments that emphasize adaptation speed over the theoretical possibility of superior ultimate performance.

Implicitly, the above statements assume that integrating elementary building blocks requires similar effort and time to the integration of larger subsystems. In practice, this is largely true but exerts its toll through inferior performance relative to what is theoretically possible. Simon's main goal is to explain why our universe is dominated by systems exhibiting *hierarchical structure in a loose sense* – which Köstler calls holarchies to distinguish from strict hierarchies (typical for rigid manmade artifacts).

Köstler makes a first attempt at characterizing these suitable subsystems, which have to survive the dynamics of the environment better and longer than the overall system (as illustrated in the watchmakers parable). Köstler calls this ability to survive the autonomy of the subsystem or Holon, whereas the contribution of the subsystem to the overall system is called the cooperativeness of the Holon. The autonomy gives the Holon the capacity to cope with changes, uncertainty and disturbances in its environment. In contrast, a subsystem developed in the context of a top-down development activity often is highly constrained in its ability to function outside the specific settings foreseen by its developers. In Köstler's view, such subsystems lack autonomy and therefore are unable to survive in a demanding and dynamic environment. More recent insights in complex adaptive systems [3] reveal however that Köstler failed to discover more insightful aspects of these stable subsystems from which almost every large, complex system in our world is built. These aspects are discussed below.

3 Holonic Systems and Complex Adaptive Systems

In [3], Waldrop describes research activities and insights on complex adaptive systems mainly originating from the well-known Santa Fé Institute. Among the many research results, two properties of complex adaptive systems are relevant for the discussion in this paper: autocatalytic sets and lock-in.

3.1 Autocatalytic Sets

Autocatalysis is more important than autonomy for the emergence and survival of systems, which eventually become subsystems in a larger system. Autonomy and adaptability are only secondary properties of autocatalytic sets, needed to maintain and increase autocatalysis in a dynamic and changing environment.

The concept of an autocatalytic set serves to enhance the probabilities in the standard biologist's theory that life emerged by chance when energy pulses strike a pool filled with organic molecules. Unfortunately for this standard theory, the smallest life forms still are so big that combining its basic molecules by chance is as likely to happen as 'a group of monkeys typing Shakespeare's oeuvre by pure coincidence.' These calculations change however drastically if combinations of molecules are formed into sets that are catalysts for themselves. If energy pulses arrive at a sufficiently high frequency, the autocatalysis implies that the pool rapidly becomes filled with members of such autocatalytic sets (exponential growth until raw material becomes scarce).

The omnipresence of such set members also means that they become the building blocks for larger molecular combinations, amongst which the autocatalytic set members will dominate again. This can be repeated until life forms emerge. If this theory is correct, the dominating life forms should be members of autocatalytic sets themselves. Mice, rabbits, weeds and insects all provide strong empirical evidence supporting the theory. Autonomy, adaptability, manipulating the environment, etc. are secondary properties of the more complex life forms (including humans) that mainly increase the intensity of the autocatalysis in favor of the own set.

To translate the above to the domain of holonic and multi-agent systems, it is necessary to identify the relevant autocatalytic sets for manmade artifacts and software systems in particular. These sets are not situated in artificial worlds inside computer platforms serving to investigate artificial life. The relevant autocatalytic sets comprise both software and humans (i.e. software users and developers). Successful software systems belong to two kinds of autocatalytic sets:

- *The economic set.* Successful software represents economic value to its users and thus mobilizes the economic means for software developers to maintain, adapt and enhance this software.
- *The information feedback set.* Successful software attracts users providing feedback on its shortcomings and merits. This information, in combination with the economic means, allows the developers to maintain, adapt and enhance the software such that it remains competitive.

The above implies that (software) system designers have to account for more than just the technical dimension to be successful. Sufficient users (and their payments) and sufficient diversity in the user community (and its feedback), relative to the competitive pressures, are necessary for emergence and survival.

Most importantly, such successful members of autocatalytic sets are the (only) systems that may become the subsystems in larger more sophisticated systems. Ceteris paribus, software systems with the intrinsic ability to serve more users or a more diverse community of users will have an edge over the competition since their autocatalysis is stronger. Note that software quality and functionality levels are likely to improve significantly through the above types of autocatalysis (personal experience with systems experiencing low levels of autocatalysis has provided the authors with strong empirical evidence supporting this statement).

3.2 Lock-In

The previous section depicts how positive feedback is instrumental in structuring worlds such that larger and more complex systems emerge and survive. This section introduces a negative aspect of such positive feedback: *lock-in into early solutions*.

Systems, in which autocatalysis can occur, may evolve along multiple trajectories where the selection amongst these trajectories strongly depends on which autocatalytic process kicks in the earliest. Since it is an exponential process, the autocatalytic set rapidly exhausts the available 'raw material' effectively eliminating the opportunity for other autocatalytic processes sharing 'material requirements' to start at all. The first process to start generally ends up dominating its world until it fails to survive the dynamics of its environment. At most, a short time window will be available for competing and faster autocatalytic processes to overtake a competitor that started earlier.

In the world of high-tech human-made artifacts, the first product to adequately serve important user requirements typically captures the market and prevents superior solutions from emerging. A well-known example is the VHS standard, being the most inferior technology amongst the competitors at the time. Making the situation worse, lock-in is actively used by commercial organizations to lock out competition. The main mechanism counter-acting lock-in consists of the dynamics of the overall system, making the dominating autocatalytic set obsolete (e.g. DVD overtaking VHS).

Relevant for holonic systems development is the poor level of suitability and adaptability of the available systems (members of autocatalytic sets) from which larger systems have to be developed. Today's systems are developed with specific user requirements in mind, and the world locks into those early solutions. Those early solutions incorporate many design choices that prevent the creation and maintenance of other and larger systems at some later time. Alternative, the later usage for somewhat different purposes of those early systems is much less effective and has lower performance than theoretically possible.

Examples of lock-in in industry are the CNC architecture of machine tools, offering virtually no responsiveness to sensor read-outs and the PLC architecture characterized by poor scale-ability to larger applications. Among others, these early designs consume the available training and support resources blocking out more advanced alternatives. Mainstream computing architectures have comparable lock-in equally reflecting their history (no hard real-time, query-and-answer functionality…). A major difference is that mainstream computing enjoys a more dynamic environment that breaks the lock-in more frequently in comparison to the industrial IT environment. Multimedia en entertainment provide the resources and drive to make mainstream computing escape from obsolete historical design choices whereas industrial IT typically has to work around its legacy. Mainstream computing is likely to conquer industrial IT, which is virtually frozen in those locked-in states.

Overall, lock-in is desirable when it simplifies the world by reducing the options and alternatives that have to be taken into account. Lock-in is undesirable when a dominating system incorporates highly unfortunate design choices, which unfortunately are commonplace since being first (and just good enough in the short run) is more important than being well designed. However, if a developers community is aware of this issue and knows methodologies to handle the lock-in problem better, the

expectation of increased returns from the better designs are likely to prevent really poor designs for achieving autocatalysis to the point that are never developed at all. Hence, the issue addressed in section 4 is how to develop and design systems, without significantly more effort, that are better suited for later usage in other systems and/or larger systems.

4 Designing for the Unforeseen and the Unknown Application

Köstler was overly flattering for the subsystems in Holonic systems. Autonomy is neither omnipresent nor decisive. Autonomy helps a subsystem to adapt and be a competitive candidate to become a cooperative subsystem (holon) within a larger system (holon). Thus, autonomous subsystems probably are more common than others but their autonomy is not the decisive element. Autocatalysis dominates to the point where the larger systems are highly sub-optimal (e.g. incorporate rigid CNC controls) or built from smaller subsystems (e.g. low-level single-axis actuator controls and sensors) decreasing adaptation speed significantly and increasing maintenance efforts.

Systems that become subsystems at some later stage(s) are built with some particular usage in mind. In today's society, virtually every system is developed for incorporation in a particular overall system. The issue addressed below is how to design and develop such systems that they are also suitable subsystems in other systems and better suited for future usage, especially when this extra comes virtually for free (i.e. by replacing arbitrary design choices with purposeful ones).

This section first presents a formal framework, defining problems and their solutions. Next, the reuse of earlier solutions is discussed within this framework. Then, design principles and guidelines are revealed. Finally, sample robust designs for partially unknown requirements are presented and discussed.

4.1 Problems and Solutions

This section formally addresses what constitutes a problem and its solution(s). The purpose of the formal approach is to present ideas more precisely. The formal approach does not produce any calculus on problems and solutions, nor does it claim completeness in a philosophical sense. The formalism mainly serves to avoid ambiguity and to delineate the ideas more sharply than would be possible in natural language.

A problem P is defined as follows:

$$\text{A problem } P \text{ is a constraint on the state space } \mathbf{U} \text{ of the universe } U, \quad (1)$$
$$\text{defining a set } \mathbf{P} = \{\, \mathbf{u} \in \mathbf{U} \mid \mathbf{u} \text{ satisfies } P \,\} \subseteq \mathbf{U}.$$

When U is the world in which we live, $\mathbf{U}$ is an infinite state space. Every state $\mathbf{u} \in \mathbf{U}$ has a time coordinate $\mathbf{t_u} \in \Re$. By definition, *reachable states at a given time coordinate* are states that either have been or can become the actual state of the universe at this given time coordinate. This universe is subject to the laws of nature (constraints), which limit the number of states that are reachable. These laws of physics imply that there is exactly one reachable state $\mathbf{u}$ for every $\mathbf{t_u} \leq \mathbf{t_{now}}$.

The universe U follows a trajectory through its state space as time progresses. This trajectory is defined for states up to $\mathbf{t_{now}}$. It consists of the states in which the universe has been in the past. The future trajectory is only partially defined. This future trajectory is constrained by physical laws, possibly including stochastic aspects, and is affected by the actions of the human and other entities in this world. These actions affect the choice of the successor states of the current state corresponding to $\mathbf{t_{now}}$. Normally, any significant impact on the trajectory requires sustained action during a substantial amount of time. A problem P is solvable by an agent (human or otherwise) if the agent is able to make this trajectory stay within the given subset **P**.

A solution S to a basic problem P is defined as follows:

Solution S of a problem P is a constraint on the state space **U** of universe U (2)
defining a set $\mathbf{S} = \{ \mathbf{u} \in \mathbf{U} \mid \mathbf{u}$ satisfies $S \}$, where
$\mathbf{S} \subseteq \mathbf{P} \subseteq \mathbf{U}$ and $\forall\, \mathbf{t} \in \Re, \exists\, \mathbf{s} \in \mathbf{S}$: $\mathbf{t}$ = timeCoordinateOf(**s**).

Agents (human or otherwise, intentionally or unintentionally) solve a given problem P when they confine, through their actions, the trajectory of the universe U to the corresponding subset **P**. Therefore, their actions – combined with the laws of the universe – correspond to constraining the state of the universe to a subset **S** of **P**. **S** cannot be empty; it must always have at least one state for every time coordinate. If S fails to comply with this condition, the agents failed to solve the problem.

Typically, **S** and **P** will differ significantly concerning the states with a time coordinate that is smaller than, equal to, or marginally larger than $\mathbf{t_{now}}$. The problem P is only concerned with what is needed/useful/… Therefore, it allows as many states as possible as long as the choice amongst them does not matter – note that this discussion only considers intrinsic aspects and makes abstraction of issues concerning the explicit specification of constraints.

In contrast, the solution S is embedded in the universe, which allows only a single state for every time coordinate in the past (including the present) and imposes severe limitations on what states can be reached in the immediate future from the current state. In other words, **S** will be significantly smaller than **P**, especially concerning states close to the present time and older. Therefore, problem-solving agents have to make choices whenever deadlines approach.

In real life, a problem solving activity consists of a sequence of actions over time. Using the above definition, such sequence of actions corresponds to a sequence of solutions $S_1, S_2, \ldots S_{end}$ that solve P, where $\mathbf{S_{end}} \subset \ldots \subset \mathbf{S_2} \subset \mathbf{S_1} \subset \mathbf{P}$. This reflects that the agents make more and more choices as time progresses in order to solve the problem and comply with the laws of the universe.

An example of the introduction of constraints can be observed in the design of a railway system to solve transportation problems: when the moment of actual usage approaches, the designers have to make more and more choices. For instance, they must select a specific value for the space in between the rails. In fact, the whole problem solving activity can be seen as a sequence of design choices, starting from the

selection of a rail-based system over other possibilities. This introduction of constraints by the solution is the key issue.

4.2 Reuse of Solutions for Unknown Problems

As explained above, complex artifacts (solutions) are constructed using members of autocatalytic sets, which are solutions originally developed to solve other problems. In other words, a solution to an overall problem must be realized by integrating existing candidate sub-solutions (a holonic system is created through aggregation of suitable smaller holons) where these candidates have been developed without knowledge about this overall problem. This section formally addresses this solution reuse process.

Formally, agent **x** solves problem P through solution S_p. Other agents solve problem Q through solution S_q. This results in the state sets $\mathbf{S_p} \subseteq \mathbf{P} \subseteq \mathbf{U}$ and $\mathbf{S_q} \subseteq \mathbf{Q} \subseteq \mathbf{U}$. For instance, agent **x** has constructed the railway system in France to answer the need for transportation. The other agents implemented similar railway systems in the remainder of Europe. These systems are defined as to include the human organization that operates, maintains and adapts/expands the hardware therein. In this example, the overall problem of providing transportation all over Europe – i.e. problem T – is to be solved by integration of the national railway systems – i.e. integration of S_p and S_q into solution S_t. Intrinsically, agents must solve problem T, where $\mathbf{T} \subseteq \mathbf{P}$ and $\mathbf{T} \subseteq \mathbf{Q}$. Practically, agents must integrate the existing sub-solutions into their overall solution; formally: $\mathbf{T} \subseteq \mathbf{S_p} \cap \mathbf{S_q}$. Indeed, society is unable to duplicate the effort of developing their national railway systems to provide an international one. Instead, it must reuse the existing systems, including their ability to adapt, to create the international connections amongst the national systems and obtain a system that transports goods and passengers across the borders in Europe.

The main problem with the creation of such solutions is that the integration fails to deliver good performance. In the example, it is relatively easy to provide international transport at a much-reduced level of service; goods and passengers need to change trains at almost every border. Higher levels of service require significant efforts: locomotives that support multiple power supply systems, railway equipment supporting variable width of the railway tracks... The sub-solutions, which were developed independently, have made mutually incompatible design decisions and these decisions have accumulated significant inertia (i.e. it has become costly to undo these decisions). Formally and with an ambitious overall problem T_A, "$\exists$ t $\in \Re$, $\forall$ s $\in S_p \cap S_q \cap T_A$: t $\neq$ timeCoordinateOf(s)" expresses that, in practice, reuse of sub-solutions cannot be reconciled with an ambitious overall problem T_A that requires a high-performance solution, as if the sub-solutions were redeveloped with problem T_A in mind.

In practice, a solution for T_A cannot readily reuse the available (partial) solutions offered without undoing a lot of design choices. Typically, society only receives a reduced level of service (i.e. solutions to easier problems), and will only gradually outgrow the old designs when technology progresses sufficiently to introduce a new solution from scratch (e.g. high-speed trains). The key issue is the introduction of

constraints that are absent in the corresponding problem and that may cause future integration problems. More precisely, it is the accumulated inertia – i.e. the effort necessary to undo such harmful design decisions – that constitutes the problem.

4.3 Designing for Unknown Problems

From the above, it becomes clear that the problem-solving agents must design a solution S_p capable of surviving in an uncertain environment concerning its future. Formally, such uncertain environment corresponds to $\wp$, a set of subsets of the state space $\mathbf{U}$. The actual future will offer an intersection of members of $\wp$ as the space that is available for S_p to contribute its part to the overall solution. Some members of $\wp$ correspond to the constraints imposed by the future problems for which solution S_p may be part of the overall solution. Alternatively, members of $\wp$ correspond to the constraints imposed by other candidate sub-solutions that may contribute to solving the bigger problem at hand. A designer of solution S_p does not know which members of $\wp$ will be present, and must try to avoid conflicts with any constraints that might be presented by members of $\wp$.

In this context, design decisions can introduce two types of constraints: stable and unstable. Stable constraints will be present in all conceivable future situations within the scope of S_p. For instance, an agent may assume that power supply will be 240V/50Hz or 130V/60Hz when designing an electrical appliance. Formally, no member of $\wp$ will be in conflict with the stable constraint. In contrast, unstable constraints represent conflicts with some members of $\wp$. Design decisions that introduce unstable constraints reduce the future capability of the solution. For instance, the designer of a rocket inertial navigation program may choose to limit the range of the acceleration supported to the limits of the current rocket, causing the crash of the first rocket of the next generation when the constraint remains undetected.

Based on the above, design principles P1 and P2 emerge. Note that these principles apply when designing lasting artifacts, like infrastructures, and not for the short-time solutions for the immediate future.

P1: Problem solvers must avoid introducing potentially harmful constraints. This guideline is twofold. First, it advises against the introduction of (any) constraints. As seen in section 4.1, this cannot be avoided for the immediate future. However, concerning problem solving in a slightly more distant future, systems implementing low and late commitment strategies avoid the introduction of constraints. This is a relatively well-known fact, and it is widely adhered to. Unfortunately, cost and complexity of such solutions seriously restrict the applicability of this design principle. For instance, building locomotives and railway wagons for variable-width tracks is prohibitively expensive/complex. Software related to the railway system has less difficulty to avoid commitment to a specific track width; it simply becomes a system parameter to which the remainder of the software must consistently adapt.

Second, the guideline encourages introducing constraints unlikely to cause future conflicts; these are called *stable constraints or decisions* in this manuscript. In short, the introduction of stable constraints simply reflects the fact that the constraint is

already present in the environment; a stable design decision preserves and reflects the scope of the problem domain. In a way, these stable constraints can be seen as an extension of the laws of nature for the application domain at hand.

P2: Problem solvers must avoid/reduce the inertia build-up for potentially harmful constraints. Again, this guideline is twofold. First, designers are encouraged to implement low-inertia versions of their solutions first and only to implement the real system when, hopefully, most conflicts among design choices have been undone in this low-inertia version of the design. This is a well-known principle and specific technologies have been developed to support this: CAD, CAE, … Indeed, it is significantly less costly to undo a design choice on a CAD drawing of a bridge than to do the same modification on the bridge itself at a later stage. Likewise, virtual reality technology is used on a regular base to discover and undo conflicting constraints in factory designs. Similarly, in the research on multi-agent manufacturing control, control software prototypes will be connected to emulations of the underlying manufacturing system first, allowing adaptation of both the control system and the underlying system in the virtual world before committing in the real world.

Second, designers must avoid reinforcing unstable constraints introduced by earlier unstable design decisions. Designers cannot avoid introducing unstable constraints, e.g. selecting a width for the railway track, when the time for implementing the solution approaches. However, the fact that this width needs to be fixed before the first locomotive can be build does not imply that this width needs to be fixed in other situations, e.g. being hard-coded in a piece of embedded software, where the cost of keeping your options open is negligible. Whereas stable constraints can be reinforced without problems – the environment or general scope of the problem domain already imposes such constraints – an earlier unstable constraint gains inertia every time another design decision introduces it again. For instance, it is a known problem that highly automated production facilities are hard to adapt because their control systems (= software) reinforce the existing way of operating and require significant maintenance for every change in the system. If the software for a control system expects a tree-shaped bill-of-materials, the introduction of disassembly operations will be problematic, especially when the software has a functional decomposition design. Ironically, the technology that is seen to be extremely flexible causes the rigidity (because of its poor design).

In summary, the novel principles for the designers are: (1) designers must prefer stable design decisions and (2) earlier unstable design decisions are no justification for later decisions imposing the same constraint(s). The first design principle avoids the introduction of new constraints. The second avoids the build-up of inertia for unstable constraints that were introduced earlier; every unstable design decision must be justifiable by itself. Earlier unstable design decisions only affect which later unstable design decisions are taken (i.e. compatible with these earlier ones), not whether a later unstable decision will be taken at all. Next to these two items, some better-known principles – low and late commitment, autonomous systems, and low inertia implementations – remain valid. Note that low commitment includes the ability to fall back on interim designs at stages before unstable constraints are introduced.

4.4 Sample Robust Designs for Unknown Problems

Although the scientific and engineering communities are largely unaware of this, our society has successfully created and developed subsystems that are well prepared for the unknown, or perhaps more precisely, are usable in a wide range of applications. This section discusses examples with varying degrees of success (or compliance with the above design principles), the key being not to rely on the wrong assumptions for a subsystem to be a valid (part of a) solution.

Physics
Physical science is an extreme example of compliance with the above design rules. At a first glance, it does not make any choices at all – apart from representation/language/symbol aspects. Looking more closely, it is possible to distinguish an applicability range. For instance, Newton's classical mechanics become invalid when velocities approach the speed of light or when dimensions become extremely small.

Science uses the ultimate source of stable constraints: the world itself. A collection of artifacts – in this case scientific theories – reflecting parts or aspects of the real world will not conflict within the scope in which they are valid. If conflicts occur, these conflicts are caused by flaws in the artifact, not by unstable design choices. Indeed, it suffice to look at the world – by means of suitable experiments – to know which theory is correct (or to discover that all are flawed).

Navigation and Transportation
Consider the following types of artifacts: route descriptions, traffic regulations and maps. Maps clearly comply with the design principles, assuming they reflect relatively long-lived aspects of some part of the world. Note that they benefit from the same source of stable constraints as scientific theories – the real world. When two maps contradict each other, at least one map contains a mistake and verifying against reality suffices to solve the conflict. Notice how maps covering overlapping sections of the world and representing overlapping aspects (e.g. tourist maps, traffic maps, military maps) can be used inside a single larger system/application. Maps augmented by an organization (including humans) to keep these maps up-to-date at adequate frequencies represent an even better solution in line with the design principles.

Route descriptions are less robust artifacts in two ways. First, route descriptions are exposed to highly unstable user requirements (origin, destination, route attributes), whose number of possible combinations is practically unlimited. Second, route descriptions are fragile with respect to sensing accuracy, disturbances and changes. When its user confuses an exit of a parking lot with a side street, route descriptions often provide a poor level of service. An even worse problem occurs when the route, indicated in the description, is temporarily blocked by road works. Traffic regulations are possibly even worse. Resolving arbitrary choices is a main reason for their existence, and it should be no surprise that combining two regulations, which came independently into existence, more or less results in undoing half of the design choices.

Software Design
Today, two major paradigms for software development exist in practice. Historically, top-down functional decomposition preceded object-oriented design. Functional design starts from functional user requirements and decomposes these in a top-down fashion. User requirements are notoriously unstable. As a consequence, subsystems

developed by functional approaches are likely to incorporate a multitude of unstable design choices. They are the analogue of the route descriptions above. Remark that they do not only share the presence of many unstable design choices (a negative aspect) but also require roughly the least effort possible to answer their original user requirements (a positive aspect).

In contrast, object-oriented design includes the elaboration of an essential or conceptual model of the area of interest [4][5]. In an object-oriented design of a navigation application, this essential model would comprise a map of the area in which the navigation occurs. In personnel administration applications, the essential model would reflect the possible life cycles of personnel. Again, the real world serves as a source of stable constraints. As with maps in navigation, the elaboration and implementation of such essential models often requires too much effort for single-use applications. However, the object-oriented approach rapidly becomes superior when user requirements are partially uncertain and the application needs to survive beyond the immediate future.

Multi-agent Systems

Often, the analytical minds of DAI researchers have been attracted to 'consolidated user requirements addressed by a MAS solution'. As a consequence, much MAS research starts from the top-down functional decomposition paradigm and fails to develop suitable subsystems from which larger systems can be composed. Symptomatic for this phenomenon are researchers calling out for solutions that are as small as possible, focusing on decision-making, automatic generation of MAS, methodologies to keep development efforts small and short.

In contrast, a number of researchers are investigating agent systems in which agents reflect part of the relevant reality: "agents must be things" [6]. This is especially true for the MAS designs that are bio-inspired [7] and/or searching for emergent functionality and self-organizing applications. When selecting these parts of the world that are to be reflected by the agents in a MAS, it is important to safeguard the possibility for strong autocatalysis (i.e. user community must be able to support and maintain the software artifact). The PROSA architecture is an example of a design aimed at maximizing the user communities of the agents therein [8].

5 Conclusion

This paper discusses the fundamental nature of holonic systems. Limited rationality implies that sophisticated systems in a dynamic and demanding environment mainly consist of large subsystems [2]. Candidate systems, available to become such subsystems, belong to two types of autocatalytic sets [3], and therefore are predetermined to a significant extent. This limitation often results in poor performance for large systems when judged against the performance that is theoretically achievable with custom-made subsystems. The latter would be obsolete by the time they are available or fail to achieve adequate autocatalysis because the existing systems have drained the world of the suitable resources. This paper analyses the root cause of this sub-optimal performance and derives design principles and guidelines to remedy this situation.

Acknowledgements

This paper presents work funded by the Research Council of the K.U.Leuven – Concerted Research Action on Agents for Coordination and Control.

References

1. Köstler, A.: The Ghost in the Machine. ARKANA S. (1990).
2. Simon, H. A.: The Sciences of the Artificial. MIT Press, Cambridge Mass. (1990).
3. Waldrop, M.: Complexity, the Emerging Science at the Edge of Order and Chaos. VIKING, Penguin group, London.(1992).
4. Cook S., Daniels J.: Designing Object Systems. Prentice Hall, London. (1994).
5. Jackson, M.: Software requirements and specifications. Addison-Wesley, (1995).
6. Parunak, H.V.D.: "Go to the Ant: Engineering Principles from Natural Agent Systems", Annals of Operations Research, 75:69-101, (1997).
7. Grassé, P.P.: La theorie de la stigmergie: essai d'interpretation du comportement des termites constructeurs, Insectes Sociaux 6, (1959).
8. Van Brussel, H., J. Wyns, P. Valckenaers, L. Bongaerts, P. Peeters: Reference architecture for holonic manufacturing systems: PROSA. Computers In Industry 37, 255-274, (1998).