# Expectation-Oriented Analysis and Design*

Wilfried Brauer[1], Matthias Nickles[1], Michael Rovatsos[1], Gerhard Weiß[1], and Kai F. Lorentzen[2]

[1] Department of Informatics, Technical University of Munich,
80290 München, Germany
{brauer,nickles,rovatsos,weissg}@cs.tum.edu
[2] Department of Technology Assessment, Technical University of Hamburg,
21071 Hamburg, Germany
lorentzen@tu-harburg.de

**Abstract.** A key challenge for agent-oriented software engineering is to develop and implement *open* systems composed of interacting *autonomous* agents. On the one hand, there is a need for permitting autonomy in order to support desirable system properties such as decentralised control. On the other hand, there is a need for restricting autonomy in order to reduce undesirable system properties such as unpredictability. This paper introduces a novel analysis and design method for open agent-oriented software systems that aims at coming up to both of these two contrary aspects. The characteristics of this method, called EXPAND, are as follows: (i) it allows agents a maximum degree of autonomy and restricts autonomous behaviour only if necessary (ii) it uses system-level *expectations* as a key modelling abstraction and as the primary level of analysis and design; and (iii) it is sociologically grounded in Luhmann's systems theory. The application of EXPAND is illustrated in a "car-trading platform" case study.

## 1 Introduction

As new requirements arise from the increasing complexity of modern software systems and from the distributedness of today's information economies, it has been recognised that the modularity and reusability provided by object-oriented analysis and design is insufficient and that there is a need for encapsulation of more functionality at the level of software components. Agent-oriented approaches [5] offer an interesting perspective with this respect, as they view *interaction* as the primary abstraction for future software engineering approaches [10, 15]. However, interaction among autonomous entities implies contingencies in behaviour since, in the most general case, neither a peer agent nor a designer can *know* what goes on inside a (semi-)autonomous agent. These contingencies are a source of potential unpredictability and undesirable chaotic behaviour at the system level. The designer is thus faced with an apparent dilemma – to apply methods for building systems for such open, heterogenous and unpredictable

* This work is supported by DFG (German National Science Foundation) under contracts no. Br609/11-1 and MA759/4-2.

domains as the Internet on the one hand (which offer new perspectives on computation and have already proven to yield more open, flexible and adaptive software) and to harness the potential dangers of openness on the other. In order to get rid of this aspect of agent-based systems, the usual design strategy is to restrict oneself to *closed* systems (see e.g. [17, 18]). This obviously means losing the power of autonomous decentralised control in favour of a top-down imposition of social regulation to ensure predictable behaviour.

To build truly *open* systems [9] means constructing systems that may be entered by autonomous entities. Even if a developer of an open system would be willing to severely restrict autonomy (and thus to give up potential advantages induced by autonomy such as decentralised control and self-organisation), it is most unreasonable to assume that full control over autonomous entities being parts of the system (temporarily or permanently) can be guaranteed under all circumstances. Taking autonomy seriously means to accept that any "strictly *normative*" (in the sense of action-prescribing) system-level design of social activity must be abandoned – instead, desired or persistent interaction patterns can only be modelled as *descriptions* of possible or probable behaviour which might or might not occur in actual operation. Likewise, *agents* can only use models of interaction as *expected* courses of social action that are always hypothetical unless when actually enacted by them and their co-actors. A combination of *normative* and *deliberative* motives in agents' actions (the former resulting from previous system behaviour, the latter from agents' autonomy) [4] makes certainty about future interactions impossible.

Starting from these observations, this paper identifies a novel level of analysing and designing agent-based software: the *expectation level*. An analysis and design method called EXPAND ("EXPectation-oriented ANalysis and Design") is introduced that uses expectations as the primary modelling abstraction and that supports the evolutionary identification, evaluation and adaptation of system-level expectations. The core idea underlying EXPAND is to make expectation-level knowledge about the social behaviour of agents *explicit* and *available* to the system analyst and designer as well as to the agents contained in the system. From the point of view of an analyst and designer, this method offers the possibility of developing and influencing open systems that consist of black-box autonomous entities which can *not* be controlled completely; and from the point of view of the agents, it allows to retain a high degree of autonomy by using the system-level expectations as a valuable "system resource" for reducing contingency about each other's behaviour. To our knowledge, EXPAND is the first analysis and design method that aims at tackling the expectation level of agent-oriented software systems explicitly and systematically. In doing so, its main contributions lie in (i) separating social expectations (which, in the most general sense, can be seen as communication structures) from agents' mental processes, (ii) in viewing expectations as *systemic* (supra-individual) data structures that are subject to manipulation by system-level software components, and (iii) in focusing on principles and techniques for analysing and designing expectations in the process of developing open agent-based systems. Another

characteristic of EXPAND is that it possesses a strong sociological background; more specifically, its underlying view of expectations and sociality follows the *systems theory* by the sociologist Niklas Luhmann [14].

The remainder of this paper is structured as follows. Section 2 presents the generic conceptualisation of expectations underlying EXPAND. Section 3 describes EXPAND, and shows how a feasible and adequate incremental analysis and design process can be derived that exploits the importance of the expectation level in open and autonomous agent-based software systems. This is followed by an exemplification of the usefulness of our approach in a case study based on a "car-trading platform" application scenario in Section 4. Finally, Section 5 provides more general considerations on the challenge of engineering agent-oriented software, shows relationships to other methods and approaches, and indicates directions for future research.

## 2 Expectations

A major consequence of the autonomous behaviour of agents is that a certain agent appears to other agents and observers more or less as a *black box* which cannot fully be predicted and controlled. Because only the actions of an agent in its environment can be observed, while its mental state remains obscure, *beliefs* and *demands* directed to the respective agent can basically be stylised only as mutable *action expectations* which are fulfilled or disappointed in future agent actions. This suggests that it is justified, and even inevitable, to integrate expectations as a modelling abstraction into the analysis and design process of open agent-oriented software. A theory that is particularly well suited for putting this suggestion into practice is Luhmann's systems theory [14]. This theory not only provides a strong notion of expectations and their role in societies, but also focuses on interaction and communication and thus on basic ingredients of agent-oriented systems. In the following, several aspects of this theory essential to EXPAND are described; this is done in more detail because these aspects are not "common knowledge" in the field of computational agency.

### 2.1 Sociality and Communication

Because we are focusing on systems with multiple inter-operating agents, we are primarily interested in expectations addressing agent interactions which constitute *sociality*: if it comes to an encounter of two or more agents, the described situation of mutual indeterminacy is called *double contingency* [14]. To overcome this situation, that is, to determine the internal state and future behaviour of the respective other agent and to achieve reasonable coordination (including "reasonable" conflicts), the agents need to *communicate*. A single communication is the whole of a message act as a certain way of telling (e.g., via speech or gesture), plus a communicated information, plus the understanding of the communication attempt. Communication is observable as a course of related agent interactions. Because communications are the only way to overcome the problem of double

contingency (i.e. the isolation of single agents), they are the basic constituents of sociality and they form the *social system* in which the communicating agents are embedded. EXPAND adopts this view, and assigns communication a key role in analysing and designing systems composed of interacting software agents.

### 2.2 Expectation Structures and Structure Evolution

As action expectations are related to communications and thus to sociality, *social structures* can be modelled as *expectation structures*. Systems theory distinguishes four concepts that correspond to expectation structures: (i) *social agents* as sets of all current action expectations regarding single physical agents; (ii) *roles* as "variables" that are associated with certain kinds of expected behaviour and that can be instantiated by different physical agents; (iii) *social programs* as flexible interaction schemes for multiple interacting social agents and/or roles; and (iv) *social values* as ratings of expected generalised behaviour (e.g., "Conflictive behaviour is always bad"). The focus of EXPAND is on social programs, since they are particularly well-suited for describing processes that occur between agents.

By processing existing expectations, agents determine their own actions, which, then, influence the existing expectations in turn. So communication is not only **structured** by individual agent goals and intentions, but also by expectations, and the necessity to test, learn and adopt expectations for the use with future communications. The process of continuous **expectation** structure adaption by means of interaction (or communication, respectively) is called *structure evolution*. As described in Section 3, this kind of evolution also plays a key role in EXPAND.

### 2.3 System-Level Expectations and System Design

Expectations regarding agent behaviour can be formed not only by other agents (as an aspect of their mental state), but also by observers with a global view of the multiagent system. Such ***system-level expectations*** are called *emergent* if they are formed solely from the statistical evaluation of the observed communications.

As we will see in the next section, system-level expectation **structures** can be used as the target of a multiagent system design and analysis process, because they allow us to observe and structure the multiagent system at the *system level* (i.e. the level of sociality and thus of communication) itself, and not just at the level of single agents as usual. But despite their supra-agent nature, system-level expectations need to be *expected* themselves by the agents to be able to have any influence on the system. The establishment of such *"expectations of expectation"* can be achieved through the communication of the **system-level** expectations to the agents or through the publishing of the **expectations** via an appropriate agent-external instance within the multiagent system. Once achieved, agents can "expect" what "is expected by the social system". As described in Section

3, EXPAND technically realises this achievement through a so-called "social system mirror".

### 2.4 Four Key Attributes of Expectations

Systems theory reveals multiple attributes of expectations, and among them the following four are of particular relevance to EXPAND: strength, normativity, representation, and derivation.

**Strength and Normativity.** Expectations can be weighted in two complementary ways, namely, w.r.t. their *strength* and w.r.t. their *normativity* (or inversely, their *adaptability*). The strength of an expectation indicates its "degree of expectedness": the weaker (stronger) the strength of an an expectation is, the less (more) likely is its fulfilment (violation). Against that, the normativity of an expectation indicates its "degree of changeability": the more normative (adaptive) an expectation is, the smaller (greater) is the change in its strength when being contradicted by actual actions. With that, the strength of a lowly normative expectation tends to change faster, whereas the strength of a highly normative expectation is maintained in the longer term even if it is obviously inconsistent with reality (i.e. with the agents' actual activities). The idea of expectation weighting based on strengths and normativity is adopted by EXPAND, and in accordance with systems theory it is also assumed that there is a continuous transition from weak to strong strength and from low to high normativity. Here are some examples of examples of extreme combinations of strength and normativity: *rules that govern criminal law* (strong/non-adaptable: even hundreds of actual murders will not alter the respective laws, and most people think of murder as a rather exceptional event); *habits* (strong/adaptable: before the times of fast food, people took full service in restaurants for granted, but as fast food became popular, they were willing to abandon this expectation); *public parking regulations* (strong/hardly adaptable: almost everyone surpasses them even if they are, in principle, rigid); and *shop clerk friendliness* (weak/adaptable: most people expect bad service but are willing to change their view once encountering friendly staff).

**Representation.** To design multiagent systems at the expectation level, we need data structures for the representation of expectation structures and statistical methods to derive emergent expectation structures from observed communications. Settling on particular representation formalisms naturally affects the level of abstraction and with it the scope of designed expectation structures. Here, we focus on social programmes, for which a basic graphical notation is introduced as illustrated in the example shown in Fig. 1. The nodes correspond to message/speech acts that are uttered and addressed to/by instances of roles ($r_i$) (i.e. agents). The directed arcs represent the respective expectation that a communication is followed by a certain subsequent communication. Arcs are labeled by pairs $s{:}n$ of real values ranging between 0 and 1, where $s$ denotes the strength

of the expectation (in Fig. 1 additionally visualised through arrow thickness) and $n$ denotes its normativity. Note that outgoing edges of a node always do have the same normativity, because the degree of change always uniformly applies to the entire distribution of their strength.

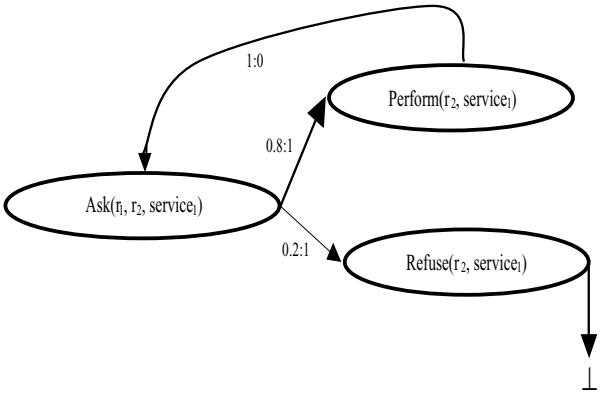


**Fig. 1.** A social program.

**Derivation.** The process by which expectation structures are derived must be able to calculate the expectations' strength and normativity values. This calculation can be done using standard statistics and probability theory. As expectations are extrapolations of observed communication processes into the future, their normativity can be quantified as the percentage of their change in the case of being contradicted by actually occurring events. For this reason it is reasonable (i) to derive the strength of an expectation by means of computing the *probability* of the communicative actions that fulfil or contradict this expectation given prior communication, and (ii) to capture the normativity of an expectation by *rules* for updating its strength.

## 3 EXPAND

Based on the description of EXPAND's sociologically founded view of expectations, this section presents EXPAND – its software-technical concept of a social system mirror and its analysis and design phases – in detail.

### 3.1 The Mirror Concept

The activities of identifying, evaluating and adapting system-level expectation structures are crucial to EXPAND. EXPAND supports these activities by means

of a so-called *social system mirror*, henceforth briefly called *mirror*. (Details on the mirror concept are provided in [13].) Conceptually, a mirror is a software component (corresponding to an EXPAND-specific CASE tool) which models an agent-oriented software system as a social system. Technically, a mirror is a knowledge base which derives system-level expectation structures from communications and makes them available to both the participating agents and the designer of the software system. The mirror has three major purposes:

1. monitoring agent communication processes,
2. deriving emergent system-level expectation structures from these observations, and
3. making expectation structures visible for the agents and the designer (the so-called *reflection effect* of the mirror).

It is important to see that not all structures that are made visible to the agents need to be emergent and derived through system observation. Rather, the mirror can be pre-structured by the designer to "reflect" manually designed ("manipulated"), non-emergent expectation structures as well. In both cases, the agents can access the mirror's and actively use the expectation structures provided by it as "guidelines" influencing their reasoning and interactivity.

For example, agents can participate in social programs which seem to be useful to them, or refrain from a certain behaviour if the mirror tells them that participation would violate a norm. Social programs (or structures in general) in which agents continue to participate become stronger, otherwise weaker. (The degree of change in strength depends on the respective normativity.) Thus, the mirror reflects a model of a social system and makes it available to the agents. As a consequence, the mirror *influences* the agents – very much like mass media do in human society. Conversely, the mirror continually observes the actual interactions among the agents and adopts the announced expectation structures in its database accordingly. In doing so, the mirror never restricts the autonomy of the agents. Its influence is solely by means of providing information, and not through the exertion of control.

The mirror, and with it actually the entire EXPAND method, realise the principle of *evolutionary* software engineering [1, 16]. More precisely, within the overall EXPAND process (i.e. within the EXPAND phases described below) two mirror-specific operations are continuously applied in a *cyclic* way:

1. it makes the system-level expectations derived by the designer from her design goals explicit and known to the agents; and
2. it monitors the system-level expectation structures which emerge from the communications among the software agents.

These two operations constitute the core of the overall analysis and design process, and together they allow a designer to control and to influence the agents' realisation and adoption of her specifications.

### 3.2 A Note on Scalability

Depending on the application, the number of agents within the multiagent system may be very large. Usually, CASE-tool supported software engineering methods like EXPAND are intended primarily for large-scale applications, and in particular in open, web-based multiagent systems (like the application described in our case study in section 4) it is undesirable to impose limitations regarding the number of participating agents. Given large numbers of interacting agents, there may be a vast amount of communication which has to be observed and evaluated by the social system mirror. Undoubtedly, the efficient achievement of this task constitutes a serious technical challenge, and a possible way to cope with this problem is to structure the mirror into – interacting – subsystems. (How such a structuring could look like in detail is an open issue that we are planning to investigate in a future work.)

On the other hand, a mirror (as an "information spreading medium") can provide for stronger coherence of the multiagent system because it makes global social structures (especially norms) known to the agents. Such structures would otherwise be invisible to the agents which usually perceive the interactions in their local social environments only. This consequence of the mirror's reflection effect is important particularly in large-scale, open and heterogenous systems where agents can tend to be isolated or to be contained disjoint social groups. In such systems, many communications are solely required to gather information about social facts, and communications can be inefficient or redundant due to ignorance of socia structures. A mirror can reduce the amount of such avoidable communications and thus is expected to have a strong ordering influence which increases the efficiency of large multiagent systems and thus can have a strong "ordering" influence on such multiagent systems.

Therefore, we expect the issue of efficiently observing and processing all communication reduce itself to a mere technical problem, while the actual *complexity* and *variety* of the ongoing communication will be effectively *reduced*.

### 3.3 The Engineering Phases

**Phase I: Modelling the System Level.** In the first phase, the software designer models the system level of the multiagent system according to her design goals in the form of design specifications which focus on "social behaviour" (i.e. desired courses of agent interaction) and "social functionality" (i.e. functionality which is achieved as a "product" of agent interaction, such as cooperative problem solving) in the widest sense (we don't take into account "second-order" design goals like high execution speed or low memory consumption). For this task, the usual specification methods and formalisms can be used, for instance, the specification of desired environment states, constraints, social plans etc. In addition or as a replacement, the specification can be done in terms of system-level expectation structures, like social programs.

**Phase II: Deriving Appropriate Expectation Structures.** In the second phase, the designer models and derives system-level expectation structures from the design specifications and stores them in the social system mirror. If the design specifications from phase I are not already expectation structures (e.g., they might be given as rules of the form "Agent $X$ must never do $Y$"), they have to be transformed appropriately. While social behaviour specifications are expectation structures *per se*, social functionalities (for instance: "Agents in the system must work out a solution for problem $X$ together") possibly need to be transformed, most likely into social programs. Sometimes a full equivalent transformation will not be feasible. In this case, the designer models expectation structures which cover as many design requirements as possible.

System-level specifications can be modelled as adaptable or normative expectations. The former can be used for establishing *hints* for the agents which are able to adapt during structure evolution, the latter for the transformation of *constraints* and other "hard" design requirements into expectations.

Figure 2 shows the spectrum of system-level specifications and expectation structures that result from this phase of the analysis and design process.

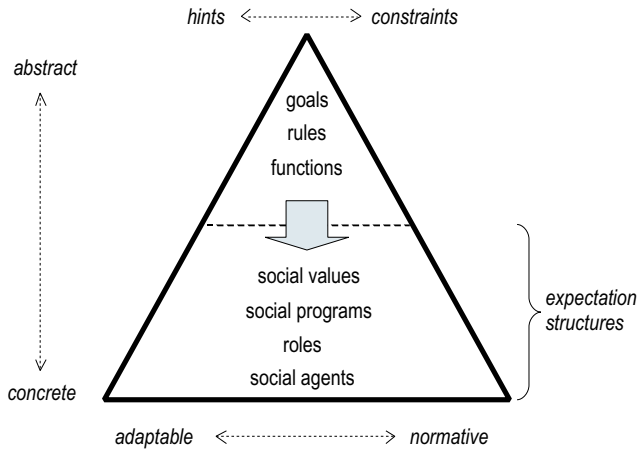


**Fig. 2.** System-level specification.

**Phase III: Monitoring Structure Evolution.** After the designer has finished modelling expectations, she makes them visible for the agents via the social system mirror and puts the multiagent system into operation (if it is not already running). In the third phase of the design and analysis process, it is up to the designer to observe and analyse the evolution of expectation structures which becomes visible to her through the mirror (Fig. 3). In particular, she has to pay

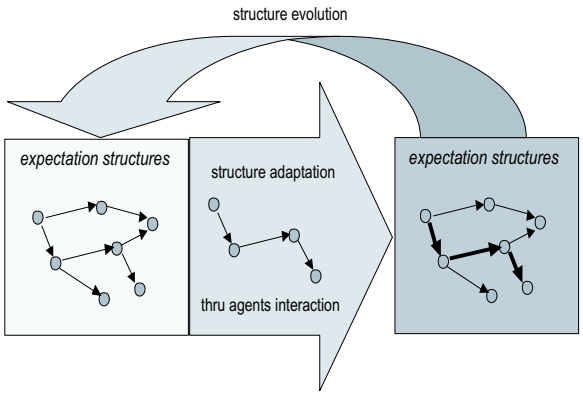


**Fig. 3.** Evolution of expectation structures.

attention to the relationship of the continuously adapted system-level expectation structures and her design specifications from phase I, which means that she analyses the expectation structures with regard to the fulfilment of norms established by the designer and the achievement of the desired social functionality. Because the mirror is only intended to show expectation structures, it could be necessary to support the mirror with a software for the (semi-)automatical "re-translation" of expectation structures into more abstract design specifications like social goals.
As long as the expectations structures develop in a positive way (i.e. they match the design goals) or no emergent structures can be identified that deserve being made explicit to improve system performance, the designer does not intervene. Otherwise she proceeds with phase IV.

**Phase IV: Refinement of Expectation Structures.** In the last phase, the designer uses her knowledge about the positive or negative emergent properties of the multiagent system to improve the system-level expectation structures. Usually, this is achieved by removing "bad" expectation structures from the mirror database, and, if necessary, by introducing new expectation structures as described in phases I and II. In addition, expectation structures which have proven to be useful can be actively supported, e.g. by increasing their expectation strength and/or their normativity. The process proceeds with phase III until all design goals are achieved or no further improvement seems probable.

Having described how the EXPAND process is supposed to be carried out in theory, we next turn to a concrete case study that will illustrate what the process might look like in practice, what problems and questions it raises, and what perspectives it offers. The phases that our analysis and design method consists of can be integrated into a single process model that is summarised by

the scheme shown in Table 1 which makes the individual actions taken in each phase more explicit.

# 4 Case Study: The Car Trading Platform

## 4.1 Scenario Overview

Imagine a website that brings together car dealers, private pre-owned car sellers and potential buyers who trade cars online (cf. `www.imotors.com` , `www.autoweb.com`, `www.autointernet.com`, `www.autotradecenter.com`). There is an "offers" section in which sellers can display images, technical details and prices of cars for sale. In the "requests" area, buyers can post requests for cars that they would be interested in. A forum is available, in which inquiries can be placed, discussions, bargaining and negotiations may take place publicly or privately (as forum users wish), etc.

## 4.2 Making Top-Level Design Decisions

Having made a decision on taking an agent-based approach for the above application, the designer must develop a top-level description of the system which will, to the least, include decisions regarding infrastructure, interaction environment and, above all, participating agents (or agent types).
Here, we will assume that the designer of the platform is designing a semi-open system: on the one hand, the system offers user interface agents that monitor the platform on behalf of users, profile users to derive interests/needs and draw their attention to interesting information on the platform. A second, pre-built type of agents are search agents that constantly re-organise the platform's database and can search it efficiently. These can be contacted by user interface agents as well as by humans for search purposes. We assume that all interactions with these search agents are benevolent, since they are not truly autonomous (they simply execute others' requests). On the other hand, there is a number of agent types that have not been designed by the designer of the platform. There can (and should) exist human and non-human agents representing individuals or organisations that interact with the platform in a "socially" unprescribed way (only restricted by implementation-level protocols and standards, e.g. FIPA compliance). Generally speaking, these agents are black-boxes for the system designer.

Further refinement of these initial design decisions will require looking at a multitude of issues, ranging from communication facilities and standards and capabilities of in-built profiling and search agents to database models etc. For our purposes, we can restrict this identification of requirements to social level characteristics of the platform since these are the subject of the EXPAND process.

| | *Action* | *Description* | *Output* |
|---|---|---|---|
| **I** | **Model system level** | **Specify social-level requirements** | **Social-level requirements specification** |
| I.1 | Model social behaviour. | Identify behaviour requirements, i.e. desired/undesired courses of interaction. | Social behaviour specification. |
| I.2 | Model social functionality. | Identify functional requirements, i.e. desired outputs of system operation. | Social functionality specification. |
| **II** | **Derive expectation structures** | **Transform requirements from phase I into appropriate expectation structures** | **Mirror instantiation with expectation structures** |
| II.1 | Derive expectation primitives. | Define what is expected of which agent(s) under what conditions. | Expectation structure primitives. |
| II.2 | Specify expectation attributes. | For all derived structures, determine strength, normativity, representation and derivation. | Complete specification of expectation structure attributes. |
| II.3 | Instantiate mirror. | Supply mirror with representations of the defined expectation structures. | Concrete mirror data structures and processing rules. |
| **III** | **Monitor system operation** | **Observe structure evolution** | **Evaluation of emergent system behaviour** |
| III.1 | Identify emergent structures. | Spot interesting/unexpected phenomena in unfolding communication processes and emergent system characteristics. Employ statistical methods, interpret data. | Catalog of emergent structures. |
| III.2 | Evaluate emergent structures. | Categorise emergent structures according to their desirability wrt requirements identified in phase I. | Evaluation for emergent system behaviours identified in III.2. |
| III.3 | Determine next action. | Assess risk of changes and urgency of changes. If changes seem necessary, continue with **IV**; else, go back to **III**. | |
| **IV** | **Refine expectation structures** | **Determine useful modifications to mirror structures** | **Modified mirror structures** |
| IV.1 | Identify structures responsible for undesired behaviour. | Involves finding "misused" or "unused" structures, structures that are too normative or too adaptable, and missing structures that lead to chaotic interaction. | Specification of appropriate modifications. |
| IV.2 | Adjust mirror contents. | Insert/delete necessary/obsolete expectation structures or adjust existing ones according to IV.1 | Updated mirror. |
| IV.3 | Deploy changes. | Determine appropriate mode of updating the mirror without disrupting operation or causing distrust toward mirror. | Deployment of modified social system mirror. |
| IV.4 | Proceed to phase **III**. | | |

**Table 1.** Detailed view of the EXPAND process.

### 4.3 Identifying Social Level Requirements

As system-level or "social level" goals, we consider the following motives of a car trading platform (CTP) provider:

1. Maximum quality of service should be provided: the range of offered and requested cars has to be broad and their specifications must relate to their prices; the reliability of transactions must be high; trust between buyers and sellers and between all users and the platform must be at a reasonable level.
2. Transaction turnover should be maximised, because it indicates (in our example) high return on investment for the CTP provider stakeholders.
3. Traffic on the platform must be maximised, to ensure high advertisement returns.

In the following, we sketch how the EXPAND process model can be applied in the analysis and design of such a system.

The dilemma in designing the social level of such a platform is obvious: system behaviour should meet the design goals and at the same time it shouldn't compromise participating external agents' private goals by being overtly restrictive. An expectation-level model of social structures is needed to cope with this situation. We next sketch the application of the suggested analysis and design process to the CTP.

### 4.4 Implementing the EXPAND Process

**Phase I: Modelling the System Level.** In the first step the social structures are modelled in the form of design specifications. They might include the following (we use natural language for convenience and concentrate only on a few design issues for lack of space):

1. Agents committing themselves to purchase/sell actions towards other must fulfil all resulting obligations (deliver, pay, invoice etc.).
2. Unreliable behaviour induces reluctance to enter business relationships on the side of others. Fraudulence leads to exclusion from the platform.
3. Interest in offers and requests must be shown by others in order to provide motivations to keep up the use of the platform.

The first specification is very important in order to foster trust among agents in such a platform. If communication were only inducing a bunch of loose pseudo-commitments that are never kept, the CTP risks becoming a playground instead of a serious, efficient marketplace. This principle is refined by item 2: the "must" in the first rule can obviously not be deontically enforced on autonomous agents, so it has to be replaced by a "softer" expression of obligation: by specifying that unreliable behaviour decreases the probability of others interacting with the unreliable individual in the future, we provide an interpretation of the former rule in terms of "consequences". Also, we distinguish "sloppy" from "illegal" behaviour and punish the latter with exclusion from the platform, a centralised

sanction that the platform may impose. The third specification is somewhat more subtle: it is based on the assumption that agents will stop posting offers and requests, if they don't receive enough feedback. Since we have to ensure both a broad range of offers as well as reasonable traffic on the site, we want to make agents believe that their participation is honoured by others so that they keep on participating (for private buyers this might be irrelevant, since they buy a car once every 5 years, but it is surely important to have plenty of professional dealers frequent the site).

The process of specifying such possible societal behaviours should be iterated on the basis of "scenarios" for all courses of communication that are of interest and seem possible, so as to yield requirements for the social system that is to be implemented.

**Phase II: Deriving Appropriate Expectation Structures.** Clearly, the three requirements above can be analysed in terms of expectations, that is, as variedly normative, possibly volatile rules that are made known to agents and evolve with observed interaction.

The second phase of the EXPAND process consists of making these abstract requirements concrete in the form of expected communication structures. As mentioned before (see 2.4) we do not intend to restrict ourselves here to a single formalism and will only use the already introduced graph-like notation to present but a few sample structures for the above specifications. Two such expectation structures derived from the above requirements 1. and 3. are shown in Figs. 4 and 5. The first example depicts an expectation structure of an order-deliver-pay-procedure in the CTP. It encapsulates high delivery and payment expectations (i.e. high transaction reliability), but also a more specific expectation as concerns availability statements that are made by dealers: although it is equally probable that the requested car will be available upon a first order, it is highly unexpected that a car that had not been available is suddenly available upon a second, identical order (in our model, responses to communication are supposed to occur in time-spans that are much shorter than those needed to change stock). Thus, the first response is given much more weight, and a notion of "honesty" in responding to orders is assumed. The second example is closely related to design goal 3 introduced above. Here, the expectation structure is used to express that few posted offers go unanswered by interested customers, and that the enquiries of such customers are responded to with high probability. By using such a structure, the designer can reassure both dealers and customers that it is *worthwhile* posting orders and enquiries to orders. If followed by the users of the CTP, such a structure would imply that postings will be answered even if the other party is not *actually* interested in the offer/question, and is just replying out of a sense of "politeness", to the end of making everyone feel that their contributions are honoured. Associated with such conventions would be the designer's goal to keep the CTP frequented, by presenting the social structures as open and rich.

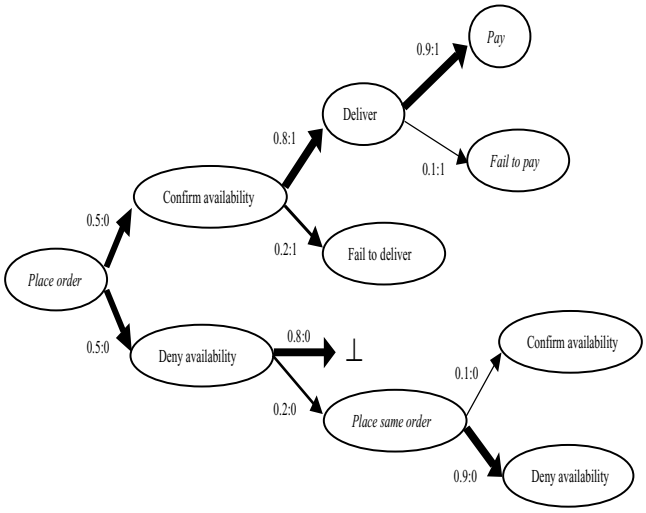


**Fig. 4.** Social program "order-deliver-pay" (buyer actions are shown in italic font, seller actions in plain face, as in all following figures, speech act arguments are omitted for lack of space): expectations about availability are balanced; in the "available" case, dealers are expected to deliver and customers are expected to pay. In the "not available" case, dealers are expected to confirm their prior statement if asked a second time (even though the probability of such a second request is low).

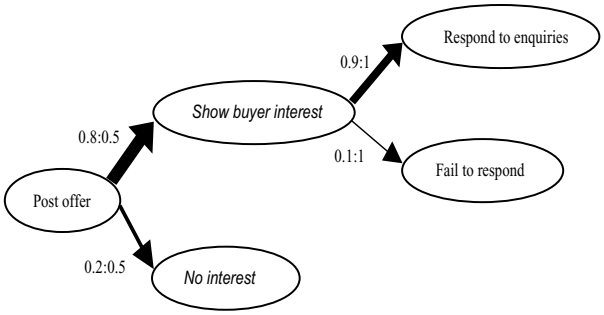


**Fig. 5.** "Initiatives are honoured" program: it is expected that dealers receive some response to their offers by potential customers, and that they react to enquiries themselves.

These simple examples given, we can return to our EXPAND design process model. We have shown how two social structure specifications were turned into concrete expectation structures (phases I and II). For lack of space, we have concentrated on *social programmes* and neglected roles, social agents and values.

Preassuming that the CTP is implemented and observed during operation, we can now proceed to phase III.

**Phase III: Monitoring Structure Evolution.** Unlike phases I and II, this phase focuses on *observation* of the system in operation in order to further refine expectation structures and their processing. It is essential to keep in mind that the systemic expectation "mirror" (as a software component) leaves plenty of choices not only as concerns the *choice* of employed expectation structures, but also with respect to how these structures are *processed*, that is, how they *evolve* through monitored agent behaviour in system operation. To stress this second aspect, we concentrate on this processing of expectations in the following examples.

Suppose, first, that we observe that actual behaviour largely deviates from that assumed in Fig. 4 in that there are many fraudulent customers who do not comply with their obligation to pay once the car has been delivered unless the dealer threatens with legal consequences several times. Obviously, identifying such a problem preassumes that interaction is tracked and that interaction patterns are statistically analysed and evaluated with respect to existing system goals. Therefore, the software engineer's primary duty is, at this stage, to spot interesting behaviours (both desirable and undesirable ones). Once realised, we are faced with a problem. By default, even though payment was designed as a norm, the "expectation mirror" would simply "truthfully" decrease the normativity in the longer term and adapt the expectation strengths subsequently so that the strength of "fail to pay" increases. This would mean that an emergent, hidden structure would be made explicit in the system, but, unfortunately, this would be a structure that embodies a functionality which does not serve the system goals (even though it has been "selected" through actual interaction) because it would make future dealers doubt the reliability of the system.

As a second example, suppose that the expectation structure in Fig. 5 corresponds to the actual system behaviour, but not because of some "polite" policy of customers to show interest in *any* dealer posting – instead, demand in cars is simply (temporarily) so high (and maybe the CTP is for some other reasons very attractive for customers) that almost no offer posting goes unanswered. Assume, further, that our initial design was to enforce "politeness" by insinuating that it was a convention of the platform, even if customers would not have been polite at all, that is, we had implemented this expectation structure as rather immutable (normativity of 0.5/1) regardless of the agents' behaviour.

In both cases, we have identified emergent (positive and negative) properties of the system that must be dealt with in phase IV.

**Phase IV: Refinement of Expectation Structures.** As designers of the platform, we can react to such emergent properties in different ways. To give a flavour for the kind of decisions designers have to make when refining expectation models, we discuss the two examples mentioned above.

In the case of the "spreading fraudulent customers", the most straightforward solution would be to impose sanctions on the fraudulent behaviour observed (i.e. to add new expectation structures). Let us assume, however, that an analysis has shown that it is too costly to verify customers' solvency and payment reserves (e.g., by inquiring other E-commerce platforms about them). On the other hand, ignoring the changes by keeping the old expectation structure (and asserting a high payment reliability in a "propaganda" way) might result in future inconsistencies: if too many individuals realise that it does not correspond to the actual social structure, they will use it less, and the "social design" level will provide lesser possibilities to influence system behaviour for the designer.

Obviously, a trade-off has to be found. One possible solution would be to extend the structure in the way suggested by Fig. 6, such that failure to pay results in reluctance of dealers to accept future orders from the unreliable customer. So, in phase IV we can specify a new functionality that feeds into the system in the next cycle.

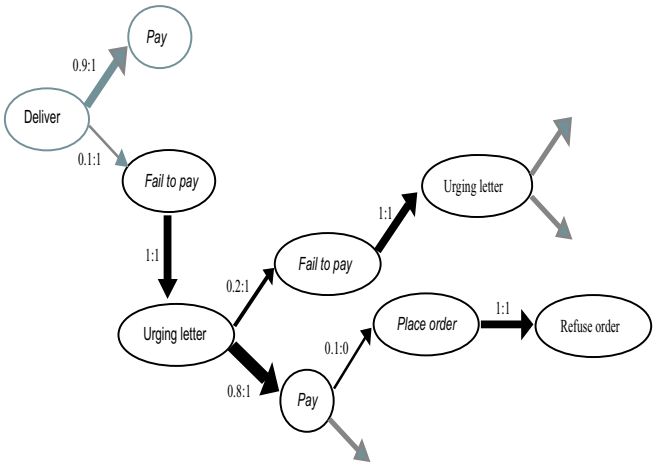


**Fig. 6.** Specifying a new functionality.

As concerns the second, "positive" emergent property, we might consider lifting the constraint of presenting an "immutable" politeness convention, in order to allow for optimisation on the agents' side: making the rule normative implies that it wouldn't change, even if, for example, dealers' offers changed over time – hence, there is little pressure for dealers to actively try to meet customer demand. Thus, if we allowed this expectation to *adapt* to the actual interest shown in offers (e.g., by updating expectation strengths as *real* probabilities, which can be achieved by decreasing the normativity value shown in Fig. 5), dealer agents would start noticing which of their postings are good (ones which increase the rate of customer inquiry) and which aren't. (After all, maximising

market efficiency in this way might help maximising CTP profits, which also depend on gross trade turnover.) We therefore decide to increase the adaptivity of this expectation structure.

Performing such modifications to the expectation level design of a system nicely illustrates how rather restrictive social structures can give way to more emergent phenomena in "safe" non-risky situations as the one depicted here when optimisation is the prominent issue, and not the reduction of chaos.

These simple examples underpin the usefulness of explicit modelling of social structures in the proposed EXPAND process model. In particular, they show how both designing social *structures* and designing the *processing* of such structures plays an important role in the open systems we envisage. Also, they illustrate the evolutionary intuition behind our design process: agents select social structures through their interaction, and designers select them through design.

## 5 Conclusions

Engineering agent-oriented software while at the same time taking autonomy as a key feature of agency seriously is a great challenge. On the one hand, it is (among other things) autonomy that makes the concept of an agent powerful and particularly useful, and that makes agent orientation significantly distinct from standard object orientation. There is an obvious and rapidly growing need for autonomous software systems capable of running in open application environments, given the increasing interoperability and interconnectivity among computers and computing platforms. On the other hand, autonomy in behaviour may result in "chaotic" overall system properties such as unpredictability and uncontrollability that are most undesirable from the point of view of software engineering and industrial application. In fact, it is one of the major driving forces of standard software engineering to avoid exactly such properties. To come up to each of these two contradictory aspects – the urgent need for autonomous software systems on the one hand and the problem of undesirable system properties induced by autonomous behaviour on the other – must be a core concern of agent-oriented software engineering, and is the basic motivation underlying the work described here.

A number of agent-oriented software engineering methods are now available (see [11] for a good survey). EXPAND is most closely related to those among these methods which also focus on the analysis and design of the system level (e.g., Gaia [18], Aalaadin [8], Cassiopeia [7], and MESSAGE [2]). All available methods as well as EXPAND aim at supporting a structured development of "non-chaotic" agent software. However, they do so in a fundamentally different way. EXPAND admits agents a maximum degree of autonomy and restricts autonomous behaviour only if this turns out to be necessary during the evolutionary analysis and design process. Against that, most other methods show a clear tendency toward seriously restricting or even excluding the agents' autonomy a priori. Different mechanisms for achieving autonomy restrictions have

been proposed, including e.g. the hardwiring of organisational structures[1], the rigid predefinition of when and how an agent has to interact with whom, and the minimisation of the individual agents' range of alternative actions. As a consequence, methods based on such mechanisms run the risk to design software agents that eventually are not very distinct from ordinary objects as considered in standard object oriented software engineering since many years. EXPAND aims at avoiding this risk by accepting autonomy as a necessary characteristic of agency that must not be ruled out headily (and sometimes even can not be ruled out at all, as is typically the case for truly open systems). With that, EXPAND is in full accordance with Jennings' claim to search for other solutions than the above mentioned restrictive mechanisms [12, p. 290]. Moreover, EXPAND with its grounding on Luhmann's theory of social systems precisely is in the line of Castelfranchi's view according to which a socially oriented perspective of engineering social order in agent systems is needed and most effective [3]. In addition to that, and more generally, this thorough sociological grounding also makes EXPAND different from other approaches that apply sociological concepts and terminology in a comparatively superficial and ad hoc manner.

Taking expectations as a level of analysis and design opens a qualitatively new perspective of agent-oriented software and its engineering. To explore and to work out such a new perspective constitutes a *long-term* scientific and practical endeavour of considerable complexity. This also is why it is not surprising that EXPAND in its current version does not yet answer all relevant issues, but necessarily includes aspects that are tentative in flavour and so leaves room for improvement. Our current work focuses on the improvement of three of these aspects that we consider as particularly important; these are a more formal treatment of system-level expectations, the technical refinement of the mirror concept, and a more systematic transition from the expectation level down to the group and agent levels and further down to the standard object level. EXPAND should be considered as a first, pioneering step toward a better understanding of the benefits and the limitations of expectation-oriented analysis and design. Faced with the challenge to build "autonomous non-chaotic agent software", we think it is important to further investigate the expectation level in general and EXPAND in particular.

## References


[1] L. J. Arthur. *Rapid Evolutionary Development: Requirements, Prototyping & Software Creation.* John Wiley & Sons, 1991.

[2] G. Caire et al. Agent oriented analysis using MESSAGE/UML. In this volume.

[3] C. Castelfranchi. Engineering social order. In *Working Notes of the First International Workshop on Engineering Societies in the Agents' World (ESAW-00)*, 2000.


[1] In [6] three organisational frameworks – markets, networks, and hierarchies – are considered, differing from each other in the degree of autonomy they concede the individual agents.

[4] C. Castelfranchi, F. Dignum, C. M. Jonker, and J. Treur. Deliberate normative agents: Principles and architecture. In *Proceedings of the Sixth International Workshop on Agent Theories, Architectures, and Languages (ATAL-99)*, Orlando, FL, 1999.

[5] P. Ciancarini and M. J. Wooldridge. *Agent-oriented Software Engineering: first international workshop (AOSE-2000)*. Springer-Verlag, Berlin et al., 2001.

[6] V. Dignum, H. Weigand, and L. Xu. Agent societies: towards frameworks-based design. In this volume.

[7] A. Drogoul and A. Collinot. Applying an agent-oriented methodology to the design of artificial organizations: a case study in robotic soccer. *Autonomous Agents and Multi-Agent Systems*, 1(1):113–129, 1998.

[8] J. Ferber and O. Gutknecht. A meta-model for the analysis and design of organizations in multi-agent systems. In *Proceedings of the 3nd International Conference on Multi-Agent Systems (ICMAS-98)*, pages 128–135, 1998.

[9] C. Hewitt. Offices are open systems. *ACM Transactions on Office Information Systems*, 4(3):271–287, July 1986.

[10] M. N. Huhns. Interaction-oriented programming. In *Agent-Oriented Software Engineering: first international workshop (AOSE-2000)*, Lecture Notes in Artificial Intelligence Vol. 1957. Springer-Verlag, 2000.

[11] C. Iglesias, M. Garijo, and J.C. Gonzales. A survey of agent-oriented methodologies. In J.P. Müller, M.P. Singh, and A. Rao, editors, *Intelligent Agents V. Proceedings of the Fifth International Workshop on Agent Theories, Architectures, and Languages (ATAL-98)*, Lecture Notes in Artificial Intelligence Vol. 1555, pages 317–330. Springer-Verlag, 1999.

[12] N.R. Jennings. On agent-based software engineering. *Artificial Intelligence*, 117:277–296, 2000.

[13] K. F. Lorentzen and M. Nickles. Ordnung aus Chaos – Prolegomena zu einer Luhmann'schen Modellierung deentropisierender Strukturbildung in Multiagentensystemen. In T. Kron, K. Junge, and S. Papendick, editors, *Luhmann modelliert. Ansätze zur Simulation von Kommunikationssystemen.* Leske & Budrich, 2001. To appear.

[14] N. Luhmann. *Social Systems.* Stanford University Press, Palo Alto, CA, 1995 (orignally published in 1984). translated by J. Bednarz, Jr. and D. Baecker.

[15] M.P. Singh. Toward interaction-oriented programming. Technical Report TR-96-15, Department of Computer Science, North Carolina State University, 1996.

[16] I. Sommerville, editor. *Software Engineering.* Addison-Wesley, 1998.

[17] M. J. Wooldridge, N.R. Jennings, and D. Kinny. A methodology for agent-oriented analysis and design. In *Proceedings of the Third International Conference on Autonomous Agents (Agents'99)*, pages 69–76, 1999.

[18] M. J. Wooldridge, N.R. Jennings, and D. Kinny. The Gaia methodology for agent-oriented analysis and design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.