Journal of Ambient Intelligence and Smart Environments 0 (2012) 1–26
DOI 10.3233/AIS-2012-0149
IOS Press

Engineering societal information systems by agent-oriented modeling

Kuldar Taveter[a,b,*,1], Hongying Du[b] and Michael N. Huhns[b]
[a]*Department of Informatics, Tallinn University of Technology, Raja 15, 12618, Tallinn, Estonia*
[b]*Department of Computer Science and Engineering, University of South Carolina, Columbia, SC, 29208, USA*

Abstract. This article is concerned with the engineering of societal information systems where technical components of a system – software agents – support the social network around which the system is centered. We propose agent-oriented modeling as a suitable software engineering approach for developing open and adaptive societal information systems. The article first outlines the steps of the software engineering process of agent-oriented modeling and shows how the resulting models can be mapped to the simulation environment. It then describes two case studies where agent-oriented modeling has successfully been applied. The first case study addresses the development of an agent-based decision-making system for helping customers in grocery shopping. The second case study treats the engineering of a societal information system for helping patients in finding healthcare providers. The simulation results from both case studies are presented and discussed. We conclude the article by comparing related work and drawing conclusions.

Keywords: Agent, multi-agent system, agent-oriented modeling, socio-technical system, societal information system

1. Introduction

This article is concerned with the engineering of societal information systems. A societal information system is based on a social network, which is a graphical structure whose nodes are roles played by individuals or organizations and whose links are specific types of dependencies among the roles. An information system should both support and. An information system should both support and take advantage of the dependencies. Examples of information systems intertwined with social networks are a system for mediating intimacy [34], a system for intergenerational play between geographically separated grandparents and grandchildren [20,33], a system for playing and sharing music based on human activities and emotions [14], and a system for emergency/disaster management [16]. Andrew P. McAfee [15] describes a corporate intranet based on a social network as "an online platform with a constantly changing structure built by distributed, autonomous, and largely self-interested peers."

Differently from the mentioned systems, we aim to design large-scale information systems that gather information from hundreds, perhaps thousands, of nodes, each associated with a person, and that affect the behaviors of the people at the nodes. In today's world this already happens by means of on-line social networking services, such as Facebook and LinkedIn. However, centralized social networking services process the information at their disposal in their own interests and share it between the participants only selectively. To alleviate this problem and improve the information shared within a large network, we are investigating the use of software agents – distributed, reactive and proactive software entities representing and working on behalf of each person in the network. Such agents gather information from individuals at the nodes of the network, process it, and enhance human behaviors at the nodes. The resulting system is a kind of multi-agent system (MAS), which can be defined as a system consisting of active and autonomous interacting agents [11,40]. The key metaphor for multi-agent systems is interaction. Mul-

* Corresponding author. E-mail: kuldar.taveter@ttu.ee.
[1] This work was performed while the first author was a Fulbright Scholar with the University of South Carolina.

ti-agent systems emphasize the design-time autonomy of the nodes and the importance of the environment in which the nodes act, which itself must often be designed [10].

The areas where the resulting information systems can help are (1) regulation (e.g., banking), (2) allocation of scarce resources (e.g., electric power, parking spaces, and emergency care), (3) distributed situation assessment (e.g., traffic jams), and (4) decentralized decision-making (e.g., finding a healthcare provider), which represent the four kinds of problems that societies confront. A *societal information system* can accordingly be defined as an information system where interactions among the participants are enhanced through their representatives – software agents – to guide their individual actions/decisions and, by aggregating local control decisions, achieve efficient and effective emergent global behavior. Agents are needed to represent the members of the society, because the computing tasks are too technical and/or too mundane and tedious to be done by individual members, while still considering the preferences of the members.

Engineering societal information systems is different from engineering other kinds of information systems. One of the reasons is that while an information system is normally seen as a system acting in the interests of enterprises or other organizations, a societal information system acts in the interests of a society as a whole, but at the same time honoring the interests of individual members of the society. As such, societal information systems are *open* systems because commuters, patients, or shoppers, for example, may join and leave the system at any time. Societal information systems are also *adaptive* systems, because they should react to their constantly changing environment, which can take the form of changes in interest rates, or in a combination of energy producers and consumers, or in traffic infrastructure. We also term societal information systems as *intelligent* systems, because they reflect the "wisdom of crowds" when recommending, for example, a healthcare provider or a grocery store.

Because of these features, developing societal information systems requires a software engineering approach different from that of engineering more conventional centralized information systems of client-server type (e.g. [1]). Because of the nature of societal information systems, it is crucial to address their design from three balanced and interrelated perspectives: information, interaction, and behavior. The information perspective matters because societal information systems are rooted in selectively sharing knowledge between software agents representing individuals. The interaction perspective is important because the very idea of societal information systems is based on interactions between software agents representing individuals in a social network. Through interactions the agents exchange knowledge but also achieve consensus by negotiations. Finally, the behavior perspective caters for considering the interests of individuals because this is what a software agent representing an individual effectively does. To sum up this paragraph, the completeness of the design process is the most important criterion for designing societal information systems.

A particular characteristic required of a suitable software engineering approach is its ability to produce efficient, intelligent, and adaptive software in a purposeful and understandable fashion. *Purposefulness* means that in light of the complexity and changing nature of the environment, it will be difficult – if not impossible – for all requirements to be stated. It is better to work at a higher level and to explain purposes in terms of goals, and, in certain circumstances, to have the system determine the appropriate path of action. By *understandability*, we mean that software should be transparent at least in its design and overall purpose.

In our view, agent-oriented modeling [28] meets well the above requirements for purposefulness and understandability and can therefore be successfully applied for developing societal information systems that are open, intelligent, and adaptive. An equally important feature of agent-oriented modeling is that rather than being yet another agent-oriented software engineering (AOSE) methodology, it provides a conceptual framework that enables to estimate the completeness of a design process. This conceptual framework – the viewpoint framework – supports the modeling of a given problem domain at three abstraction layers – analysis, design, and platform-specific design – and from three balanced and interrelated viewpoint aspects: interaction, information, and behavior. Because of this, agent-oriented modeling (AOM) is compatible with any AOSE methodology or their combination and can be applied in any problem domain. Examples of how AOM has been applied in different problem domains and with various AOSE methodologies are presented in Part II of [28]. The viewpoint framework is further described in Section 2.

Another appropriate feature of agent-oriented modeling is that it views multi-agent systems as socio-technical systems. A socio-technical system is a system containing both a social aspect and a technic-

al aspect, each of which can be treated as a subsystem [27]. The notion of socio-technical systems is very useful for understanding and defining societal information systems. A societal information system is a socio-technical system where technical components of a system – software agents – support the social network around which the system is centered.

This article addresses developing and applying societal information systems for decentralized decision-making in the problem domains of shopping for groceries and choosing healthcare providers. Because it is generally complicated to arrange experiments in a society, we have developed computer simulations for the two problem domains mentioned, using the NetLogo [38] platform of agent-based simulation. However, as most of the agent-oriented models developed in this article are platform-independent, they can also be used for developing real-life societal information systems in addition to simulated ones.

The major contributions of this article are the following:

- Describing a software engineering approach for designing societal information systems;
- Validating the approach by designing two "proof-of-concept" societal information systems.

The rest of this article is structured as follows. Section 2 gives an overview of agent-oriented modeling. Section 0 describes how the modeling constructs of agent-oriented modeling can be mapped to the programming constructs of NetLogo. Sections 4 and 5 respectively describe the application of agent-oriented modeling to the case studies of social grocery shopping and finding a healthcare provider. For both case studies, these sections also describe how agent-oriented models have been validated by NetLogo simulations. Section 0 compares related work and Section 7 draws conclusions.

2. Agent-oriented modeling

Agent-oriented modeling described in [28] is a holistic approach for analyzing and designing socio-technical systems consisting of humans and technical components, both of which are subsumed under the concept of agent. The core of agent-oriented modeling lies in the viewpoint framework that can be populated with different kinds of models. Table 1 depicts the viewpoint framework populated with a particular set of models from [28] that we have chosen for the two case studies described in Sections 4 and 5. The rationale for choosing this particular set of models is described further below in this section. The viewpoint framework represented in Table 1 maps each model to the vertical viewpoint aspects of interaction, information, and behavior and to the horizontal abstraction layers of analysis, design, and platform-specific design. Each cell in the table represents a specific viewpoint. Proceeding by viewpoints, we next give an overview of the types of models employed in this article.

Table 1
The model types of agent-oriented modeling

Abstraction layer	Viewpoint aspect		
	Interaction	Information	Behavior
Analysis	Role models and organization model	Domain model	Goal models
Design	Agent acquaintance model and interaction models	Knowledge model	Behavioral scenarios
Platform-specific design	Platform-specific design models		

From the viewpoint of *behavior analysis*, a *goal model* is a container of three components: goals, quality goals, and roles [28]. A *goal* is a representation of a functional requirement of the socio-technical system. A *quality goal*, as its name implies, is a non-functional or quality requirement of the system. Goals and quality goals can be further decomposed into smaller related sub-goals and sub-quality goals. The resulting hierarchical structure is used to show that the subcomponent is an aspect of the top-level component. Goal models also determine roles that are capacities or positions that agents playing the roles need to contribute to achieving the goals. *Roles* are modeled in detail in the viewpoint of interaction analysis [28].

From the viewpoint of *interaction analysis*, the properties of roles are expressed by role models. A *role model* describes the role in terms of the responsibilities and constraints pertaining to the agent(s) playing the role. An *organization model* is a model that represents the relationships between the roles of the socio-technical system, forming an organization [28]. The most common (and perhaps most important) relationships are control, benevolence, and peer, as conceived by Zambonelli, et al. [41]. In the *control relationship*, one role delegates responsibilities to another. In the *peer relationship*, either role can

delegate responsibilities to another. Finally, in the *benevolence relationship*, a role offers to fulfill responsibilities for another if it is in the offering role's interests.

From the viewpoint of *information analysis*, a *domain model* represents the knowledge to be handled by the socio-technical system. A *domain model* consists of domain entities and relationships between them. A domain entity is a modular unit of knowledge handled by a socio-technical system [28].

From the viewpoint of *interaction design*, an *agent acquaintance model* outlines interaction pathways between agents of the socio-technical system [28]. *Interaction models* represent interaction patterns between agents of the given types. They are based on responsibilities defined for the corresponding roles [28].

From the viewpoint of *information design*, the *knowledge model* represents private and shared knowledge that the agents need for functioning in the socio-technical system [28].

Finally, from the viewpoint of *behavior design*, we model how agents make decisions and perform activities. This is expressed by a *behavioral scenario* that describes how an agent of the given type contributes to achieving the goals set for the system [28].

As emphasized in Section 1, agent-oriented modeling is a generic approach rather than another AOSE methodology. It means that rather than using particular types of models, the completeness of the design process matters. Design is complete when all the viewpoints corresponding to the cells of Table 1 are covered by models. For example, in Chapter 7 of [28] it is demonstrated how the viewpoint framework can be populated by (combinations of) models originating in the following AOSE methodologies: Gaia [4], MaSE [6], Tropos [2], Prometheus [21], ROADMAP [12], and RAP/AOR [30].

We next describe the rationale for choosing a particular set of models described by Table 1 for designing societal information systems. First, as is shown by Table 1, this set of models results in a complete design, where all the viewpoints are covered in a balanced way. Second, according to our experience, these models are appropriate for the development of open, adaptive, and intelligent systems of the kind described in Section 1. *Openness* of systems is supported by goal models [28] that postpone deciding the system boundary until platform-independent design takes place compared with, e.g., use cases of UML [19], where the system boundary is decided at the beginning of requirements engineering. The benefit of postponing the system boundary is that it enables any role in an open system to be played by a human or software agent. For example, in the resulting simulations described in this article all the roles are played by software agents, while in a real societal information system, some roles are played by humans and other roles by software agents. *Adaptivity* and *intelligence* of systems are supported by adopting the concept of agent in designing societal information systems and software agents for implementing them, because agents are by definition reactive to changes in the environment. Agents are also proactive, that is, capable of initiating actions based on their knowledge, which can reflect the "wisdom of crowds."

Table 2
Coverage of the viewpoint framework by the software engineering process for designing societal information systems

Abstraction layer	Viewpoint aspect		
	Interaction	Information	Behavior
Analysis	Q4–Q9	Q10	Q1–Q3, Q7
Design	Q11–Q12, Q18	Q16, Q19	Q13–Q15, Q17

Subsequently we outline a software engineering process that prescribes the order in which the models of different viewpoints, such as behavior analysis models and information design models, should be created when developing societal information systems. The process consists of a set of questions that facilitate system development. The questions have been adapted and modified based on [29,39]. We next explain the modifications we introduced to similar questions pertaining to the case study of an aircraft turnaround simulator described in [29].

While the case study of an aircraft turnaround simulator described in [29] results in an agent-based simulation, just like the case studies presented in this article, the models used in [29] differ because of a different problem domain and different simulation platform. In particular, while [29] applied motivational scenarios, we have chosen to use just goal models. Also, [29] does not make use of a domain model, agent acquaintance model, and interaction models, and uses activity descriptions instead of behavioral scenarios. In addition, [29] utilizes agent models of a different kind. Considering all this, the set of questions applied in this article is different from the set applied in [29].

Table 2 shows how the questions applied by us cover the abstraction layers of analysis and design and the viewpoint aspects of interaction, information, and behavior. The abstraction layer of platform-specific design is not addressed by the questions. The

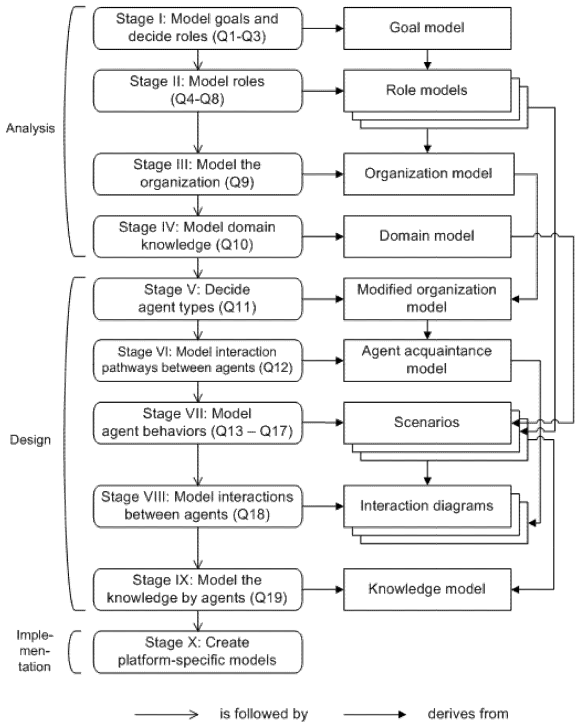

Fig. 1. The software engineering process for designing societal information systems.

reason is that we do not need any platform-dependent models because of the high abstraction level of the implementation constructs of the simulation environment employed by us – NetLogo [38]. Instead, in Sections 4.2 and 5.2 the abstraction layer of platform-dependent design is covered by explanations of developing NetLogo simulations.

The order of applying the questions represented in Table 1 is described by Fig. 1. The figure represents the stages of the software engineering process, showing for each stage the questions that it entails and the model(s) in which the answers to the questions are recorded. For example, in our case studies, the answers to the questions Q1, Q2, and Q3 are recorded in the goal model created for the system. Answering the questions represented in Table 2 and Fig. 1 produces the agent-oriented models. The questions to be asked will be presented in Section 3.1 when explaining the case study of societal grocery shopping.

3. Mapping agent-oriented models to NetLogo

In this section, we give an overview of some basic programming constructs of NetLogo and show how agent-oriented models described in Section 2 can be mapped to them. NetLogo [38] is a programmable modeling environment for simulating natural and social phenomena. System designers using NetLogo can give instructions to hundreds or thousands of agents all operating independently. This makes it possible to explore the connection between the micro-level behavior of individuals and the macro-level patterns that emerge from the interaction of many individuals. Because of this, NetLogo is a suitable environment for simulating societal information systems.

The NetLogo world is made up of agents that can follow instructions. Each agent can carry out its own activity simultaneously with the activities performed by other agents.

In NetLogo, there are four types of agents: turtles, patches, links, and the observer. *Turtles* are agents that can move around in the world. The world is two-dimensional and is divided up into a grid of patches. Each *patch* is a square piece of "ground" over which turtles can move. *Links* are agents that connect two turtles. The *observer* does not have a location – one can imagine it as looking out over the world of turtles and patches. We can also think of an observer as of a human agent in a socio-technical system.

When NetLogo starts, there are no turtles yet. The observer can make new turtles. Patches can also make new turtles. Even though patches cannot move, they are otherwise just as "alive" as turtles and the observer. Patches can be used for representing the elements of a (smart) infrastructure.

Patches and turtles have coordinates. The coordinates of a patch are always integers, but a turtle's coordinates can have decimals. This means that a turtle can be positioned at any point within its patch; it does not have to be in the center of the patch.

Links do not have coordinates; instead they have two endpoints, each of them a turtle. Links appear between the two endpoints. Links can be used for simulating communication pathways between agents.

In addition to individual agents, NetLogo allows its observer to create sets of agents – called agentsets. An agentset can contain turtles, patches, or links, but just one type at once. The elements of an agentset are in a *different* random order every time the agentset gets used.

In NetLogo, commands and reporters tell agents what to do. A *command* is an action for an agent to carry out. A *reporter* computes a result and reports it. Most commands begin with verbs ("create", "die", "jump", "inspect", "clear", etc.), while most reporters are nouns or noun phrases.

NetLogo has two kinds of commands and reporters. Commands and reporters built into NetLogo are called *primitives*. The NetLogo Dictionary has a complete list of built-in commands and reporters. Commands and reporters that the programmer can define are called *procedures,* which are named. Many commands and reporters take inputs: values that the command or reporter uses in carrying out its actions.

Agent variables are places to store values in an agent. An agent variable can be a *global variable*, a *turtle variable*, a *patch variable*, or a *link variable*. If a variable is a global variable, there is only one value for the variable and every agent can access it. Turtle, patch, and link variables are different. Each turtle has its own value for every turtle variable, and each patch has its own value for every patch variable. The same is true for links.

Some variables, such as the one representing the color of a turtle, are built into NetLogo. The observer can also define his/her own global and local variables. A *local variable* is defined and used only in the context of a particular procedure or part of a procedure. NetLogo has many built-in variables and procedures that support output to the screen and input-output from files.

NetLogo supports lists. A list lets the observer store multiple pieces of information in a single variable. Each value in the list can be a value of any type: a number or a string, an agent or agentset, or even another list.

NetLogo allows the observer to define different "breeds" or types of turtles and links. Once the observer has defined breeds, he/she can make the different breeds behave differently. There are two kinds of link breeds: breeds of directed or undirected links.

The programming constructs of NetLogo are seemingly quite different from the modeling concepts of agent-oriented modeling. However, at a closer look, the NetLogo programming constructs can be understood as the ones defining agents and their environments. As pointed out in [28], an environment can be either a real physical environment or a virtual environment. An environment simulated by means of NetLogo is an example of a virtual environment.

For example, in the context of smart mobile applications, turtles can model and simulate the owners of smartphones who go to different grocery stores or healthcare providers or who drive their automobiles in traffic. The local or private knowledge of such agents can be represented by means of turtle variables and their shared knowledge or public knowledge accessed by them – by global variables. The relationships between knowledge items are represented in NetLogo as calculations or derivations involving the respective NetLogo variables. For example, the relationship type "Is based on" between the knowledge items Fuel Cost and Route is represented as a calculation of finding the fuel cost based on the route chosen. Acquaintances (communication pathways) between agents can be modeled and simulated as links between turtles. The infrastructure, such as roads, can be modeled and simulated as a set of patches forming the environment for the agents. All in all, such a view is consistent with the one treating both agents and their environments as first-class citizens [37].

When we turn from the level of instances to the level of types, we also discover obvious mappings between agent-oriented modeling and NetLogo. For example, roles of agent-oriented modeling are mapped to agent types, which are in turn mapped to breeds of turtles. Similarly, private and shared knowledge items that correspond to domain entities are respectively mapped to local and global variables of NetLogo. The types of organizational relationships between agents, such as control, benevolence, and peer, correspond to breeds of links between turtles whereas.

Goals and behavioral scenarios of agent-oriented modeling correspond well to procedures of NetLogo that typically define the behavior of a set of turtles rather than just one turtle. The biggest disadvantage of using NetLogo for simulating multi-agent systems is that NetLogo does not directly support interactions between agents, and interactions therefore have to be implemented indirectly through the use of global variables.

A detailed mapping between the concepts of agent-oriented modeling and the programming constructs of NetLogo is presented in Table 3. In Sections 4.2 and 5.2 we employ this mapping in two case studies.

4. Case study of grocery shopping

4.1. Analysis and design

As we have previously described in [7], aided by information systems for analyzing customer buying data, supermarket chains continually alter the prices of products to maximize their profits. They do this by, in essence, experimenting on their customers. For example, the price of a product might be raised at one store until customers stop buying it. This maximum

Table 3
The mappings between agent-oriented modeling and NetLogo

Modeling concept of problem domain analysis	Modeling concept of platform-independent computational design	Programming construct of NetLogo
Role (role model)	Agent (agent model)	Turtle
Role (role model)	Agent type (agent model)	Turtle breed
Goal (goal model)	Behavioral scenario	Procedure
Domain entity (domain model)	Private knowledge item (knowledge model)	Local (to turtle) variable
Domain entity (domain model)	Shared knowledge item (knowledge model)	Global variable
Relationship between roles in a domain model (organization model)	Acquaintance (agent acquaintance model)	Link between turtles
Relationship between domain entities (domain model)	Relationship between knowledge items (knowledge model)	Calculation or derivation involving the knowledge items
Relationship type (domain model)	Relationship type (knowledge model)	Link breed

price is then used at all of the stores in the chain. The customers at the supermarkets, however, do not have any comparable information systems that might aid them in price comparisons and are often at the mercy of the stores. Most stores do not post their prices online, so that customers have to visit each store to find the prices of groceries, which makes comparison shopping prohibitive [7].

In this section we describe the design of an online system where customers could post the prices paid for groceries (this could be automated by scanning barcode labels of the products and later on by querying the RFID tags of the products) and where a prospective shopper could enter a grocery list and obtain a pointer to the store with the lowest total price. This would enable comparison shopping for groceries and would render the customer-to-store interactions fairer. It would also encourage stores to offer their true prices to avoid driving away potential customers. However, the effort required from the customers would be substantial. To make the effort reasonable and manageable, each customer could benefit from an agent that represented his/her interests and interacted with the agents of the other customers and, possibly, with store agents [7]. The highest-level goal – purpose – of such an information system is obvious and simple: "Perform shopping". We next explain the software engineering process by applying the questions represented in Table 2 to elaborate the highest-level goal "Perform shopping":

Q1 (Apply recursively to all goals starting with the system's purpose): What are the sub-goals of the given goal that are needed to achieve it?

Example: For achieving the "Perform shopping" goal, the sub-goals "Join the system", "Create shopping list", "Find potential stores", "Decide stores' shopping baskets", "Buy products", and "Exchange price and quality information" need to be achieved.

Table 4
[28] Notation for modeling goals and roles

Symbol	Meaning
	Goal
	Quality goal
	Role
	Relationship between goals
	Relationship between goals and quality goals

Q2 (For each goal of the goal tree): What are the quality goals that have to be considered when achieving the given goal?

Example: For achieving the "Exchange price and quality information" goal, the quality goals "Secure", "Minimal participation", and "Anonymous" have to be considered.

We next decide the roles by answering the following question:

Q3 (Attach to the lowest-level sub-goal possible): What are the roles that are required for achieving the goals?

Example: For achieving the "Perform shopping" goal, the roles Customer and Store are required.

The resulting *goal model* of the societal information system of grocery shopping is represented in Fig. 2. The notation for representing goals and roles is shown in Table 4, which originates in [28]. The

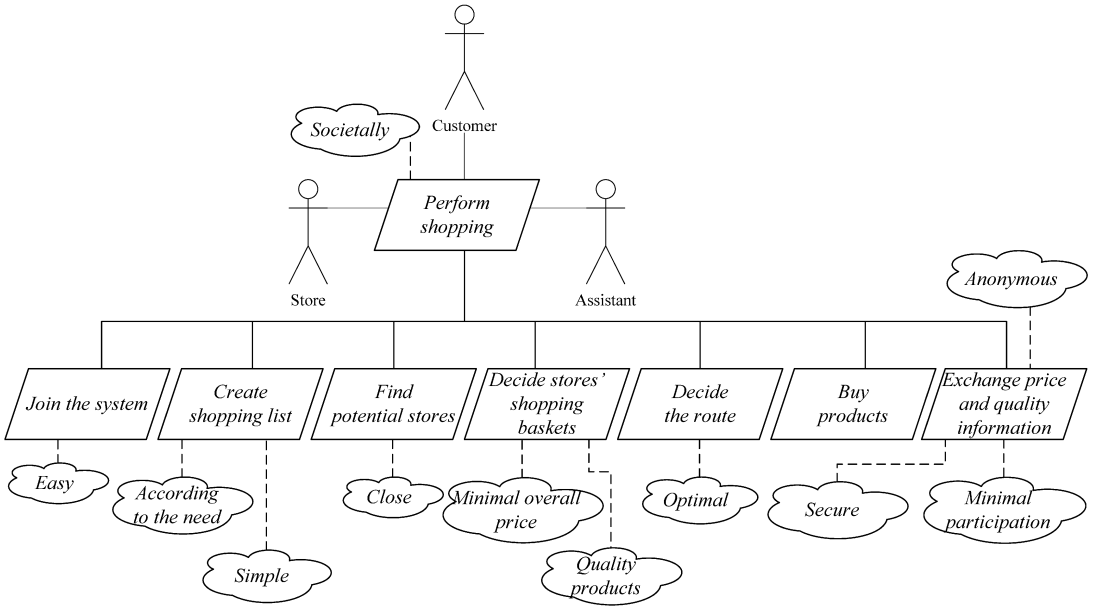

Fig. 2. The overall goal model for a societal information system of grocery shopping.

goal model is obtained as a result of applying questions Q1–Q3 recursively.

The goal model represented in Fig. 1 reflects that answering question Q2 has yielded a number of quality goals. First of all, the "Societally" quality goal attached to the highest-level goal of the information system expresses that customers are willing to share some local information, such as information about prices, to cooperate in improving the social welfare. The "Easy" quality goal pertaining to the "Join the system" functional goal states that starting using the system should not require much effort from a customer. The "According to the need" and "Simple" quality goals express the requirements for creating a shopping list with the help of the system. These quality goals should be considered when, for example, designing a user interface for the real-life societal information system. According to the "Close" quality goal, potential stores should be close to the customer. Deciding on what closeness exactly means is deliberately deferred to the design stage of the real societal information system when technical means available for determining proximity within the given information system are better understood. The "Minimal overall price" and "Quality products" quality goals represent criteria for deciding the stores and the shopping baskets of the products purchased from them. The overall price is concerned with both the cost of products purchased and the cost for fuel used for shopping. Finally, the "Secure", "Minimal participation", and "Anonymous" are quality requirements for exchanging information with other customers about prices and quality of products sold by different stores. The quality goal "Minimal participation" means that product information should be exchanged in as automated a fashion as possible. At the current stage of technological development, a complying design would translate to leaving for the customer only the operation of scanning the barcode labels of products.

Next, role models are obtained and refined by means of asking questions Q4, Q5, Q6, Q7, and Q8 presented below:

Q4: What are the responsibilities of each role that have to be fulfilled for achieving the respective goal(s)?

Example: For achieving the "Perform shopping" goal, the Customer role has the following responsibilities: "Join the system", "Create the shopping list", "Find potential stores", "Decide the stores and their respective shopping baskets", "Decide the route", "Drive to the stores", "Buy products", "Post price and quality-of-product information", "Receive price and quality-of-product information", and "Store price and quality-of-product information". Note that the responsibilities of roles are orthogonal to functional goals.

Q5: If one was to hire more staff to handle the problem, what positions would need to be filled?

Example: The problem in the given case is shopping. Some help would make shopping easier for a customer. We therefore complement the goal model with the Assistant role.

Q6 (For each new role): What responsibilities of the existing roles does the new role take?

Example: The Assistant role takes the responsibility of finding potential stores, deciding and proposing to the customer the stores and their respective shopping baskets, deciding and proposing the route, and posting and receiving price and quality-of-product information.

Q7 (For each new role): Does the new role bring along any new goals and sub-goals? What are the new responsibilities of each role that have to be fulfilled for achieving the respective goal(s)?

Example: The Assistant role brings along the goal "Create typical shopping list" and "Add a product to typical shopping list". These goals are not shown in Fig. 2. They are reflected by the corresponding responsibilities of the Assistant role and by the new responsibility "Pick products from the typical shopping list" of the Customer role.

Similarly, the metaphor of hiring new staff yields the Coordinator role that takes the responsibilities of storing and making available information about products purchased, including prices that customers have paid for groceries in different stores.

Q8 (For each role): To which social policies (rules, regulations, or codes of behavior) is this role required to adhere in order to fulfill its responsibilities successfully?

Example: To benefit from the anonymous product information posted by other customers, the customer must authorize anonymous posting of his/her product information.

Question Q3 yielded the roles Customer and Store and the Assistant role has been added as a result of asking question Q5.We represent the role models for two of the resulting roles – Customer and Assistant – in Tables 5 and 6, respectively. We do not represent the Coordinator role because of its simple nature. The organization model is then created based on question Q9:

Q9 (For each role): Which other roles does this role rely on? For each role that it relies on, what is the relationship between these roles?

Example: The customer relies on the store to buy grocery products. The customer also relies on other customers for recommendations and on the assistant for help. Consequently, the Store role is benevolent towards the Customer role; the Customer is a peer of the Customer role and controls the Assistant role.

Table 5
The role model for Customer

Role	Customer
Description	The role of customer in grocery shopping
Responsibilities	Join the system
	Create the shopping list
	Pick products from the typical shopping list
	Confirm the stores and shopping baskets suggested by the assistant
	Confirm the route suggested by the assistant
	Drive to the stores
	Buy products
	Register product information
Constraints	To benefit from the product information posted by other customers, the customer must authorize posting of his/her product information.

Table 6
The role model for Assistant

Role	Assistant
Description	The role of a customer's assistant in grocery shopping
Responsibilities	Find potential stores
	Decide and propose the stores and their respective shopping baskets
	Decide and propose the route
	Create the typical shopping list
	Post price and quality-of-product information
	Receive price and quality-of-product information
Constraints	Creating a shopping list should be simple and reflect the need by the customer
	Potential stores must be close to the customer
	The preferences by the customer must be honored when deciding the stores and their shopping baskets
	The overall price should be as low as possible
	Quality of products chosen should be as high as possible
	Informing other customers should be secure and anonymous
	To post price and quality-of-product information, the customer must have scanned or inserted the product information

Also, the Coordinator role is a peer to the Assistant role. The resulting *organization model* is represented in Fig. 3.

The following question Q10 yields the domain model:

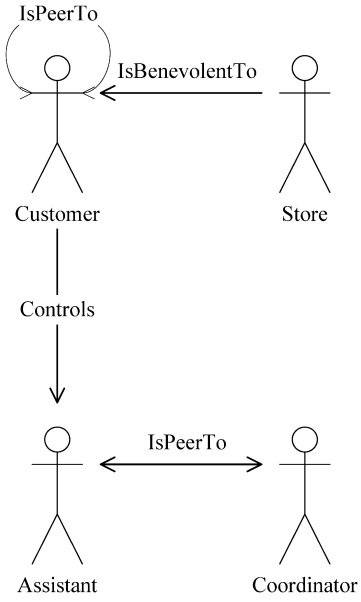

Fig. 3. The organization model of societal information system of grocery shopping.

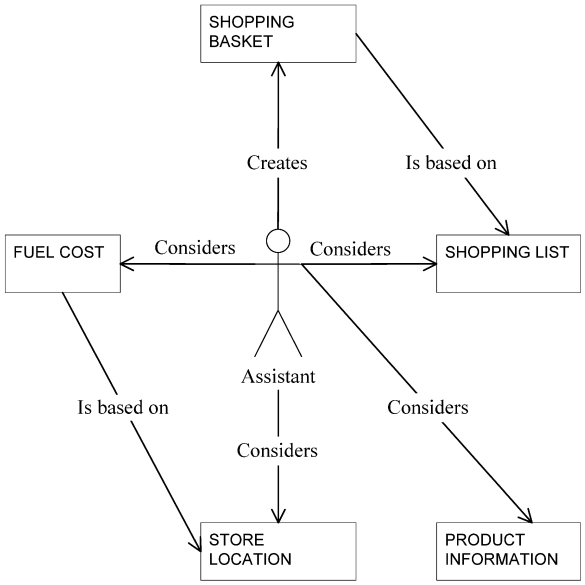

Fig. 4. Partial domain model of the societal information system of grocery shopping.

Q10 (For each role): What domain entities will this role require in order to fulfill its responsibilities successfully? What are the relationships between the domain entities, if any?

Example: To fulfill its responsibilities successfully, the Assistant needs to access the domain entities Shopping List, Shopping Basket, Fuel Cost, Store Location, and Product Information, containing the price and quality information about different products sold at various stores. A shopping list is a list of products that should be bought, while a shopping basket is a list of products that should be bought from a specific store. The resulting partial *domain model* of the societal information system of grocery shopping is depicted in Fig. 4. The domain model also identifies the relationships between the domain entities, such as "Is based on" in Fig. 4.

We next elaborate the domain model represented in Fig. 4.The elaborated domain model of the societal information system of grocery shopping is depicted in Fig. 5. The domain model shows that the Customer creates a Shopping List that is considered by the Assistant along with Fuel Cost and Product Information when creating Shopping Baskets. We can also see from Fig. 5 that Fuel Cost is based on Route, both of which are calculated by the Assistant. The Route is, in turn, based on Store Information, particularly Store Location. Figure 5 shows that a Shopping Basket consists of Products sold by a particular Store. We can also see from Fig. 5 that a Customer registers Product Information for each Product purchased by him/her and that Product Information is posted by the customer's Assistant and stored by the Coordinator. Finally, Fig. 5 represents that the Assistant creates a Typical Shopping List, which is a kind of Shopping List. Please note that some roles, such as Assistant and Customer, occur in Fig. 5 several times. This is important for just the clarity of Fig. 5.

Question Q10 completes the problem domain analysis and platform-independent design is next, beginning with question Q11:

Q11 (For each role): Is this role to be performed by a human agent, a software agent, or an external hardware/software system? Decide the agent type for each software agent.

Example: In the societal information system of grocery shopping, the Customer and Store roles are performed by human agents and the Assistant and Coordinator roles are performed by software agents. As a result, we can conclude that the *software system boundary* of the societal information system is obviously between the roles Customer and Assistant represented in Fig. 3. We can also say that on the upper side of Fig. 3 is the *human subsystem* of the socio-technical system to be designed and on the lower side is the *technical subsystem*.

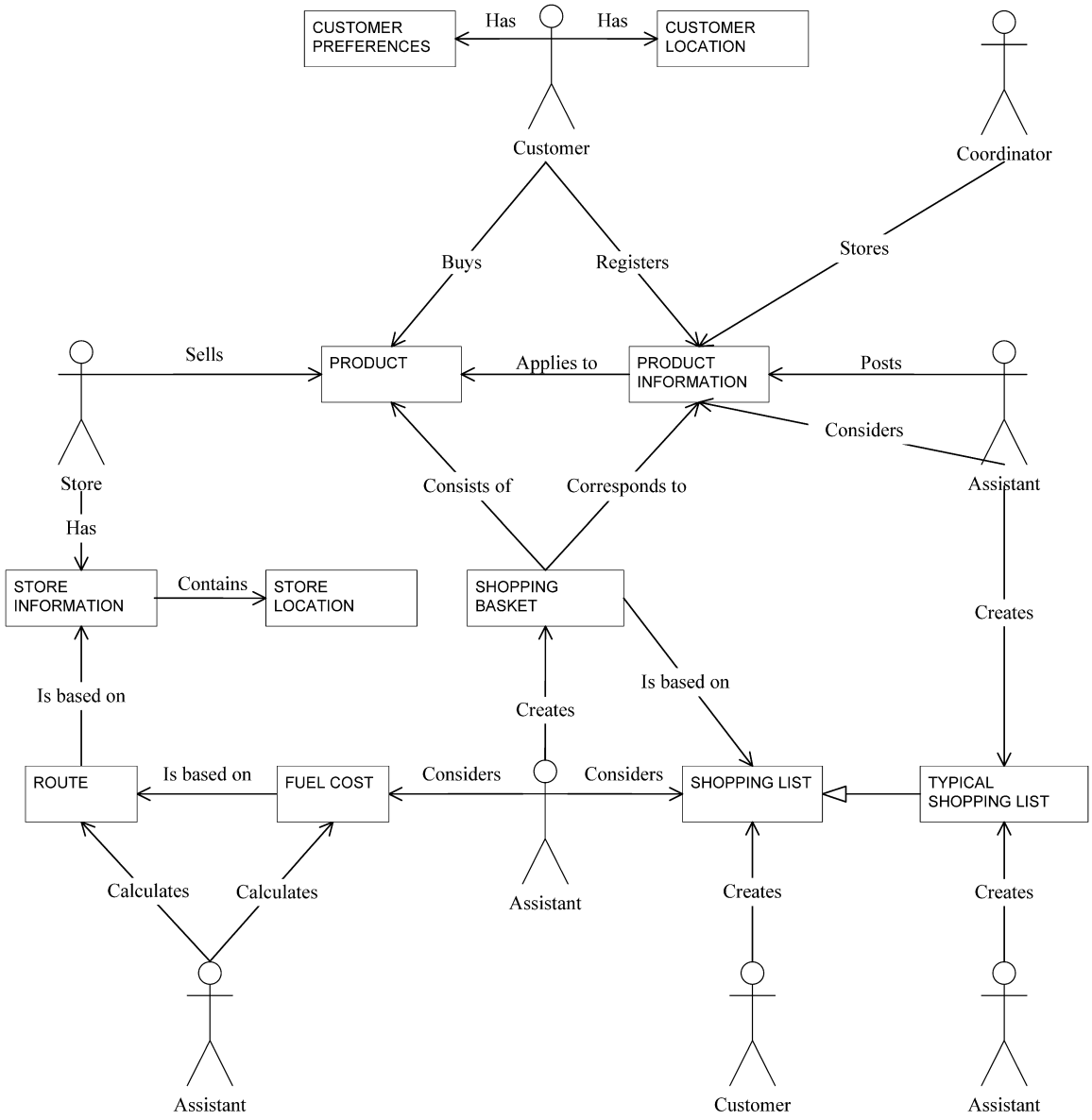

Fig. 5. The elaborated domain model of the societal information system of grocery shopping.

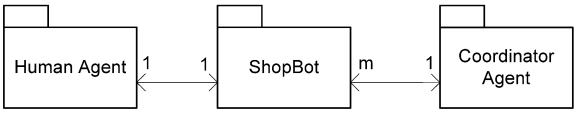

Fig. 6. The agent acquaintance model for the societal information system of grocery shopping.

The types of software agents playing the roles of Assistant and Coordinator are ShopBot and Coordinator Agent, respectively. The agent types are depicted in Fig. 6.

Q12 (For each decided agent type): With what other agents does an agent of the given type interact?

Example: An agent of the ShopBot type interacts with the Coordinator Agent that stores the information received from many customers about the prices and quality of products.

The answer is recorded in the *agent acquaintance model*. The agent acquaintance model for the societal information system of grocery shopping is represented in Fig. 6. The model reflects that each human participating in the system interacts with

Table 7
The behavioral scenario for a Human Agent playing the role of Customer

BEHAVIORAL SCENARIO 1						
Role			Customer			
Agent type			Human Agent			
DESCRIPTION						
Trigger	Condition	Step	Activity	Other roles/agent types involved	Domain entities	Relevant goals (quality goals)
A request by the customer	Alternative	1	Create a shopping list	Assistant/ShopBot	Shopping List	Create shopping list (According to the need, Simple)
		2	Pick products from the typical shopping list			
A request by the ShopBot	For each Store chosen	3	Confirm the stores, shopping baskets, and route	Assistant/ShopBot	Store Information, Shopping Basket, Product, Customer Location, Route, Fuel Cost	Find potential stores (Close), Decide stores' shopping baskets (Minimal overall price, Quality products), Decide the route (Optimal)
A request by the customer	For each Product to be bought	4	Register product information	Assistant/ShopBot	Product, Product Information	Exchange price and quality information (Secure, Minimal participation, Anonymous)

his/her ShopBot agent and with the Coordinator Agent. Please note that a rectangle in Fig. 6 is the UML symbol for component with a *different connotation*.

We next apply questions Q13, Q14, Q15, Q16, and Q17 for creating behavioral scenarios for the agent types just decided. The behavioral scenarios obtained by answering questions Q13–Q17 are contained by Tables 7 and 8 for agents playing the roles Customer and Assistant, respectively. Each step of a behavioral scenario consists of a trigger, condition, step number, description of the activity, other roles involved and the types of agents by which they are played, types of domain entities accessed by the activity, and relevant goals and quality goals from the goal model. *Trigger* is the event to which the agent reacts by starting this activity. *Condition* specifies in which order and how many times a given activity should be performed. The implicit condition is that an activity must be performed sequentially and once:

Q13 (For each responsibility of each role): What activities are required for an agent to fulfill this responsibility?

Example: An agent playing the Assistant role and fulfilling the "Decide and propose the stores and their respective shopping baskets" responsibility must perform the activity of deciding the shopping baskets, where the commodities to be purchased from each store chosen are decided. This activity is represented as step 2 in Table 8.

Q14 (For each activity): What sub-activities and atomic actions does this activity consist of and in what order are they performed?

Example: The activity of deciding the shopping baskets consists of sub-activities of deciding a shopping basket for each potential store chosen by the system. These sub-activities are performed in sequential order. For the purpose of simulation, the sub-activity of deciding a shopping basket for a store is considered as an atomic action.

Q15 (For each activity identified): What triggers this activity?

Example: The activity of deciding the shopping baskets is triggered by the completion of price and quality evaluations for the stores.

Q16 (For each activity identified): What knowledge items does this activity need to access?

Example: The activity of deciding the shopping baskets needs to access the following knowledge items: Shopping List, Product, Product Information, Store Information, and Shopping Basket.

Q17 (For each activity identified): What goals and quality goals included by goal models are relevant for successful performing of the given activity?

Example: For the activity of deciding the shopping baskets, the relevant goal is "Decide stores' shopping

Table 8
The behavioral scenario for a ShopBot agent playing the role of Assistant

BEHAVIORAL SCENARIO 2						
Role			Assistant			
Agent type			ShopBot			
DESCRIPTION						
Trigger	Condition	Step	Activity	Other roles/agent types involved	Domain entities	Relevant goals (Quality goals)
The shopping list has been created	For each product on the Shopping List	1	Retrieve the price and quality-of-product evaluations of the product in different nearby stores	Coordinator/ Coordinator Agent	Shopping List, Product, Product Information, Store Information, Customer Location	Find potential stores (Close), Exchange price and quality information (Secure, Minimal participation, Anonymous)
The price and quality-of-product evaluations have been retrieved	For each potential Store	2	Decide a shopping basket	Customer/Human Agent	Shopping List, Product, Product Information, Store Information, Shopping Basket	Decide stores' shopping baskets (Minimal overall price, Quality products)
Shopping baskets have been decided	For the best combinati on of Stores	3	Decide the route	Customer/Human Agent	Store Information, Customer Location, Route, Fuel Cost	Decide the route (Optimal)
The route has been decided		4	Propose the stores, shopping baskets, and route	Assistant/ShopBot	Store Information, Shopping Basket, Product, Customer Location, Route, Fuel Cost	Find potential stores (Close), Decide stores' shopping baskets (Minimal overall price, Quality products), Decide the route (Optimal)
Product information for the given product has been registered		5	Post price and quality-of-product information	Coordinator/ Co-ordinator Agent	Product, Product Information	Exchange price and quality information (Secure, Minimal participation, Anonymous)
Product information for the given product has been registered		6	Complement the typical shopping list		Product, Product Information	Create shopping list (According to the need, Simple)

baskets" and the relevant quality goals are "Minimal overall price" and "Quality products."

If there are any other *roles/agent types* involved, the given activity is an interaction and should also be represented as an interaction model. Obtaining interaction models, such as the one exemplified by Fig. 7, is the purpose of the next question to be asked, Q18:

Q18 (For each activity identified): Does the successful completion of this activity require other agents? If it does, what messages are involved?

Example: The successful completion of retrieving the price and quality-of-product evaluations activity modeled as step 1 in Table 8 requires interactions with the Coordinator Agent. The prototypical mes-

Table 9
The types of domain entities shared by agents of the corresponding types

	ShopBot	Coordinator Agent
ShopBot	Shopping List, Shopping Basket, Customer Location, Route	Product, Product Information, Store Information
Coordinator Agent	Product, Product Information, Store Information	

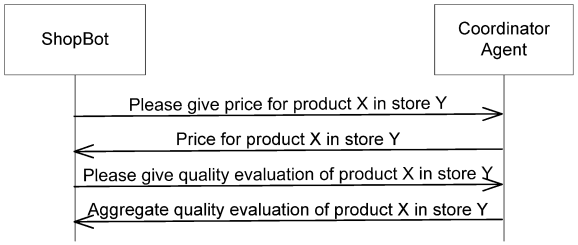

Fig. 7. A message sequence between ShopBot and Coordinator Agent.

sages involved are represented in the interaction diagram in Fig. 7.

Finally, we derive the knowledge model in the following way:

Q19 (For each activity identified): What knowledge items are shared by which agents and what knowledge items are private for which agents?

Example: The knowledge items Product, Product Information, and Store Information are shared between agents of the type ShopBot and the Coordinator Agent, while the knowledge items Shopping List, Shopping Basket, Customer Location, and Route are private for each ShopBot.

The answer is recorded in the knowledge mode represented as Table 9, which shows for each pair of man-made agent (software agent or robot) types which domain entities are shared between agents of the corresponding types and which ones are private.

4.2. Developing a simulation for grocery shopping

In Section 4.1 we described how a real societal information system of grocery shopping should be designed. As it is complicated to experiment with such information system in a society, we have to rely on simulations for evaluating our approach. We decided to perform simulations on the NetLogo environment that was introduced in Section 3. In order to implement the agent-oriented models on NetLogo, we map also roles that are normally performed by human agents, such as Customer in the example of grocery shopping, to software agent types. In addition, we assume the quantity of a specific item in a store is either zero or infinity.

To make our simulations as realistic as possible, we used data about relative importance of components in the Consumer Price Indexes by the U.S. Bureau of Labor Statistics' Division of Information Services [32].

We next describe from different viewpoints how we mapped agent-oriented models of the societal information system of grocery shopping to the programming constructs of NetLogo.

From the *viewpoint of platform-dependent information design*, we represented in the simulation the domain entities introduced by agent-oriented modeling in Section 4.1 as the following NetLogo variables:

- The Product domain entity – in terms of the product's identifier and price.
- The Shopping List domain entity – in terms of a list of product identifiers of the products that a customer wants to buy and their quantities.
- The Store Location and Customer Location domain entities – in terms of the simulated coordinates of a customer and store.
- The Shopping Basket domain entity – in terms of the store and list of products.
- The Product Information domain entity – in terms of the store, product identifier, price, and product quality.

The *viewpoint of platform-dependent behavior design* covers the behaviors of the ShopBot agents and the Coordinator Agent, as well as software agents performing the roles of Customer and Store. In accordance with the behavioral scenarios represented in Tables 7 and 8, the starting point of the simulation of societal shopping consists of a customer's shopping list of product identifiers and the quantities of the respective products.

We now take the *viewpoint of platform-dependent interaction design*. At step 1 of the behavioral scenario represented in Table 8, a ShopBot agent retrieves from the Coordinator Agent the prices and quality evaluations for the products on the shopping list in all of the nearby stores. By "nearby", we mean the stores that are within a certain range of a customer in terms of the simulated coordinates. The number of stores

within the range can be set at the start of the simulation. We set it to 12 as an overestimate of real situations. The exchange of messages to be implemented is modeled as an interaction diagram in Fig. 7. As we mentioned in Section 3, NetLogo does not support interactions between agents, and therefore interactions represented in Fig. 7 were implemented indirectly through using global variables.

We now return to the *viewpoint of platform-dependent behavior design*. At step 2 of the behavioral scenario represented in Table 8, the software agent corresponding to the ShopBot decides shopping baskets for different stores by finding for each product on the shopping list the store with the lowest and second lowest price for that product. If the prices for a product sold by several stores are the same, the agent chooses the product with the highest quality. The agent considers all possible combinations of the two prices and calculates the total cost as the sum of product prices. At step 3 of the behavioral scenario, the software agent corresponding to the ShopBot agent calculates the shortest route between the best combination of stores and the associated fuel cost, which is added to the cost of the products.

For comparison, we also calculated the overall cost if the customer chooses to go to stores using three other strategies that people often adopt in real life:

- Choose one store randomly and buy all the items at that store.
- Go to the nearest store.
- Randomly go to one of the five nearest stores.

Thereafter we calculated the ratio of the total product and fuel cost according to the three methods over that of the method of societal grocery shopping. For each parameter, such as customer location and store location, we fixed other parameters, varied the parameter in question randomly, performed the experiments 100 times and took the average as the final results. The simulation results are represented in Table 10. Each row in Table 10 represents the results after varying a specific parameter. As shown, our approach of societal grocery shopping is better than the other 3 methods for all cases, which saves 21% or more in cost. For the "vary shopping list" case, the shopping list sometimes contains fewer items, which leads to small overall savings.

We also did experiments using real prices collected from 5 stores and checked the robustness by allowing the stores to lie about prices. This article does not describe the simulation results any further because of space limitations and the interested reader is referred to [7] for more details.

Table 10
Simulation results using randomly generated price data

Simulation parameters	Mean ratio of the shopping method to the method of societal grocery shopping		
	Choose store randomly	Choose nearest store	Choose 1 store randomly from 5 nearest
Vary customer location	1.2328	1.2365	1.2178
Vary store location	1.2351	1.2325	1.2269
Vary item price	1.2150	1.2180	1.2225
Vary number of items	1.2637	1.3317	1.2911
Vary shopping list	1.1732	1.1080	1.1573

5. Case study of healthcare

5.1. Analysis and design

In the current case study, we have chosen to focus on the healthcare system of the United States. The healthcare quadruple in the United States consists of (1) patients, (2) healthcare providers (hospitals, health centers, medical laboratories, etc.) and provider networks, (3) insurance companies, and (4) the government. There are a variety of information systems available to support healthcare providers, provider networks, and insurance companies, but none to support individual patients. Because patients are naturally distributed and are typically *willing to assist each other*, societal agent-based information systems instead of centralized information systems would be appropriate for helping patients. In such systems, each patient would be represented by a software agent. The agent would assist its principal in understanding and interpreting insurance rules, finding the most cost-effective insurer, finding a good healthcare provider, providing advice on cost-effective drugs and care, and monitoring the spread of cold and flu symptoms and their treatments. Feedback and information sharing by other patients would be extensively used in such systems.

In this subsection, we focus on a particular aspect of assisting patients – finding appropriate healthcare providers. We describe how a societal information system of finding healthcare providers can be designed and simulated. We do not repeat the questions from Table 2, but instead refer to the examples of applying these questions in Section 4.1.

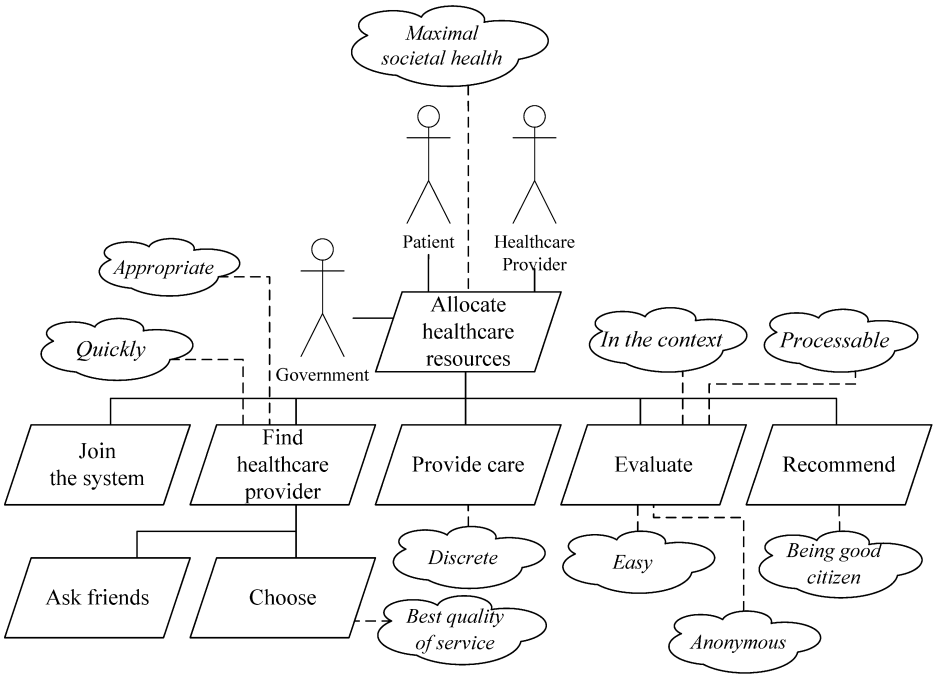

Fig. 8. The goal model of the societal healthcare information system.

Exactly as in the problem domain of grocery shopping, we start designing a societal information system for healthcare by deciding its purpose: "Allocate healthcare resources" (among the members of the society). Its realization can be viewed as a socio-technical system. We next elaborate the goal tree by responding to questions Q1 and Q2. The resulting goal model is represented in Fig. 8. The goal models reflects that patients need to join the societal information system and find a healthcare provider by it, care has to be provided, and patients have to evaluate care and recommend healthcare providers to other patients. Each of these sub-goals represents a particular aspect of allocating healthcare resources, which is to be achieved by the overall socio-technical system.

In addition to functional goals, we need a number of quality goals in the goal model. Achieving the highest-level goal "Allocate healthcare resources" is characterized by the quality goal "Maximal societal health", which determines the quality criterion according to which healthcare resources should be allocated in a society. A possible metric for this criterion is an average number of annual sick days per person in a society. We also add "Quickly" pertaining to the functional goal "Find healthcare provider". The meaning of this quality goal is obvious. In addition, we express that a healthcare provider to be found should be appropriate. In the analysis phase, we do not need to specify the precise meaning of the "Appropriate" quality goal, because it is elaborated in the phase of designing the real-life societal information system where we decide how exactly this attribute of a physician can be represented and what algorithms and software solutions are available for supporting it. However, it is highly relevant to capture this quality goal by analysis models that are used in round-table discussions between customers and other non-technical stakeholders and the developers of the societal information system.

As we plan to use social networking for finding a healthcare provider, we elaborate the "Find healthcare provider" functional goal into two sub-goals: "Ask friends" and "Choose". We characterize achieving the second of these functional goals by the "Best quality of service" quality goal, meaning that the healthcare provider who offers the best overall quality of service should be chosen. Again, we do not worry here how to measure the overall quality of service and postpone this until the design phase, where we decide technical means for supporting quality appraisals and social networking.

Achieving the "Provide care" functional goal is characterized by the "Discrete" quality goal with an obvious meaning. The "Evaluate" functional goal is modified by four quality goals. The quality goal "In the context" represents that evaluation has to occur in

the context of receiving the service, preferably before leaving the facilities of the healthcare provider or at least on the same day. This quality goal implies the need to introduce some context awareness and activity recognition (see, e.g. [18]) into the system. The "Easy" quality goal means that evaluating a healthcare provider should be easy for a patient. Potential design decisions for achieving this quality goal involve using a cell phone or a specialized device for evaluation. The "Processable" quality goal means that the evaluation should be presented in a form amenable to computer processing. What exactly it means is again left up to the design. For example, depending on the system design, it could mean that all evaluations should be expressed on a scale from 1 to 5. Or alternatively, if the system includes a data-mining component, it could mean that evaluations can be expressed in a controlled natural language. Finally, the "Anonymous" quality goal expresses that no evaluation by a patient should identify the patient. The "Recommend" functional goal is modified by the "Being good citizen" quality goal, meaning that recommending healthcare providers to other patients is a voluntary activity benefiting a society as a whole.

Having defined the goals and quality goals for the system, we now proceed to question Q3 that guides us to decide the roles that are required for achieving the goals. In this case study the roles are obvious: Patient and Healthcare Provider. Based on question Q4, we now represent each of these roles in terms of its responsibilities. There is also a third role – Government – but its modeling is not relevant for the societal information system to be designed.

Analogously to designing the societal information system of grocery shopping, some help would make finding a healthcare provider easier for a patient. We therefore complement the goal model with the new Assistant role in response to question Q5. The Assistant role takes up the responsibilities of asking friends for recommendations, choosing a healthcare provider, recommending healthcare providers, and partially evaluating the care. By this we have obtained an answer to question Q6. Differently from the societal information system of grocery shopping, the Assistant role does not bring along any new goals or sub-goals in reply to question Q7.

Finally, answering question Q8 results in a set of constraints included by role models. The resulting role models for Patient, Healthcare Provider, and Assistant are shown in Tables 11–13.

Table 11
The role model for Patient

Role	Patient
Description	The role of patient in U.S. healthcare
Responsibilities	Join the system
	Confirm or reject the healthcare provider recommended by the assistant
	Receive care
	Evaluate care
Constraints	An evaluation by the patient should consider the efficiency and quality of care
	An evaluation by the patient should be given in the context of receiving the care
	An evaluation by the patient should be available to his/her friends

Table 12
The role model for Healthcare Provider

Role	Healthcare Provider
Description	The role of healthcare provider in U.S. healthcare
Responsibilities	Provide medical service
Constraints	Medical service should be provided in a discrete manner
	Medical service should be provided as fast as possible

Table 13
The role model for Assistant

Role	Assistant
Description	The role of a patient's assistant in U.S. healthcare
Responsibilities	Ask the patient's friends for recommendations
	Choose a healthcare provider for the patient
	Recommend healthcare providers to the patient's friends
	Evaluate care
Constraints	The most appropriate and best possible healthcare provider should be chosen
	Healthcare providers should be recommended to the friends honestly based on evaluations by the patient
	An evaluation by the patient should not reveal the identity of the patient
	An evaluation by the patient should be amenable to computer processing
	An evaluation by the patient should be given in the context of receiving the care
	An evaluation should be easy to perform by the patient.

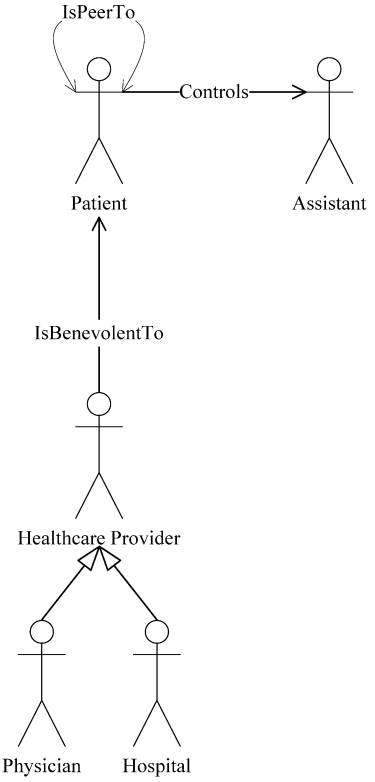

Fig. 9. The organization model of the societal healthcare information system.

We proceed with question Q9 that asks for the types of relationships between the roles. The resulting organization model is represented in Fig. 9. All three major relationship types – peer, benevolence, and control – are represented in the organization model. First, as we are addressing social networks, there is the "IsPeerTo" relationship attached to the Patient role. Second, since healthcare providers provide services to patients, there is the "IsBenevolentTo" relationship between the roles Healthcare Provider and Patient. Third, in finding healthcare providers, a patient needs help that is provided by his/her assistant. This is reflected by the "Controls" relationship between the roles Patient and Assistant. The organization model also shows that there can be different types of healthcare providers, out of which physicians and hospitals are modeled in the figure. Our design of the societal information system of healthcare will focus on patients finding physicians.

After modeling the goals, roles, and organization of the societal healthcare information system, according to question Q10 we next address the knowledge

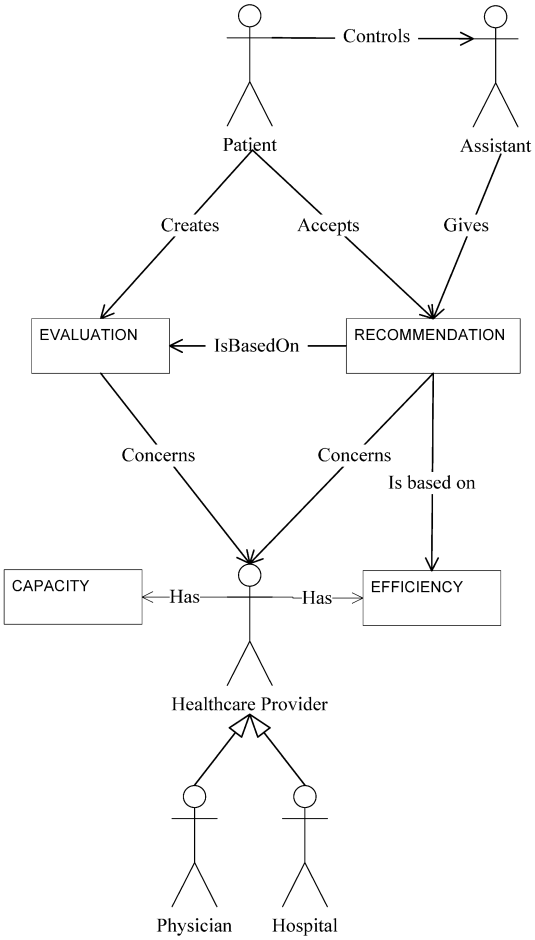

Fig. 10. The domain model of the societal information system of healthcare.

to be represented within the system. We do this by identifying the types of domain entities related to the roles. The resulting domain model is represented in Fig. 10. As each healthcare provider has predefined capacity and efficiency, we attach the Capacity and Efficiency domain entity types to the Healthcare Provider role. According to the role models represented in Tables 11 and 13, a patient evaluates a healthcare provider based on its efficiency and patients' assistants recommend healthcare providers. We accordingly place the Evaluation and Recommendation domain entity types between the roles Patient, Assistant, and Healthcare Provider. This way we obtain a domain model from the organization model.

Having created the goal model, as well as the models of relevant roles, the organization model, and

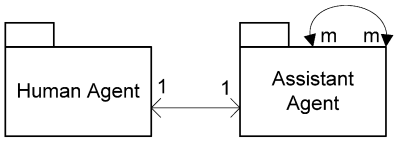

Fig. 11. The agent acquaintance model for the societal information system of healthcare.

the domain model, we have completed the analysis phase of agent-oriented modeling. We now proceed with design and decide the agent types according to question Q11. First, in a socio-technical system to be designed, the role Assistant should obviously be mapped to the Assistant Agent software agent type. Since a patient is a real human that is treated by another real human – a physician – we map both the roles Patient and Healthcare Provider to the Human Agent type. The software system boundary of the societal information system is obviously between the roles Patient and Assistant represented in Fig. 9. Regarding the Healthcare Provider role, the societal information system to be designed does not include any software agents for healthcare providers, because a societal healthcare information system aims at helping patients in the first place. However, agents assisting healthcare providers to maximize societal health as is modeled in Fig. 8 can be envisioned in the future.

The agent acquaintance model resulting from question Q12 is represented in Fig. 11, where the acquaintance links model that each patient interacts with his/her Assistant Agent and that different Assistant Agents communicate with each other. Note that in terms of interactions, the resulting solution is a pure *peer-to-peer* solution differently from the case study of grocery shopping where the resulting solution includes the Coordinator Agent.

To model the behaviors of agents of the decided types, we transform responsibilities of the roles into activities attached to the agent types. We do this by applying questions Q13–Q17. As a result, we obtain behavioral scenarios for agents playing the roles Patient, Assistant, and Physician. These behavioral scenarios are contained by the respective Tables 14–16.

The activity "Find a physician" performed by an Assistant Agent modeled in Table 15 involves interactions between software agents of patients. If a patient's Assistant Agent cannot recommend any physicians based on its principal's experience, it will turn to agents of other patients. We accordingly represent in Fig. 12 the interaction protocol between agents of the type Assistant Agent. We remind here that the difference between interaction protocol and other kinds of interaction models is that an *interaction protocol* models some aspects of the agent behaviors along with their interactions [28]. The model shows that the Assistant Agent of a patient's friend may respond with a recommendation or suggest the Assistant Agent of the friend's friend. This means that the interaction protocol shown in Fig. 12 is recursive until a pre-determined depth, which is represented by the "Loop" behavioral construct whose repeating condition is presented in the programming style. A friend's Assistant Agent may also ignore a request, which is modeled by an Option box in Fig. 12. The interaction protocol modeled in Fig. 12 constitutes a reply to question Q18 in the software engineering process.

As modeled in Table 15, the activity "Evaluate" performed by the Assistant Agent is triggered by a patient leaving the physician's office. This reflects the "In the Context" quality goal, which in Fig. 8 is attached to the "Evaluate" functional goal. How the leaving is to be perceived is left to more detailed design, which is beyond the scope of this article. A possible solution may involve the timeframe of the physician office visit in question and perceiving the geographical coordinates of the patient [18].

The behavioral scenario modeled in Table 15 also shows that "Find a physician" and "Evaluate" activities are performed sequentially. In societal information system for healthcare this is always the case, because the Assistant Agent does not perform any activities between these activities while a patient is attended by a physician.

Finally, distinguishing between private and public domain entities based on question Q19 is straightforward, because the domain entity Evaluation is private to the patient and Assistant Agent helping him/her, while the domain entity Recommendation is shared between different patients and instances of Assistant Agent. Similarly, the domain entity Capacity is private to each Healthcare Provider, while the domain entity Efficiency is shared between the physician and patients who have visited him/her. The domain entities Capacity and Efficiency form a basis for how patients evaluate healthcare providers.

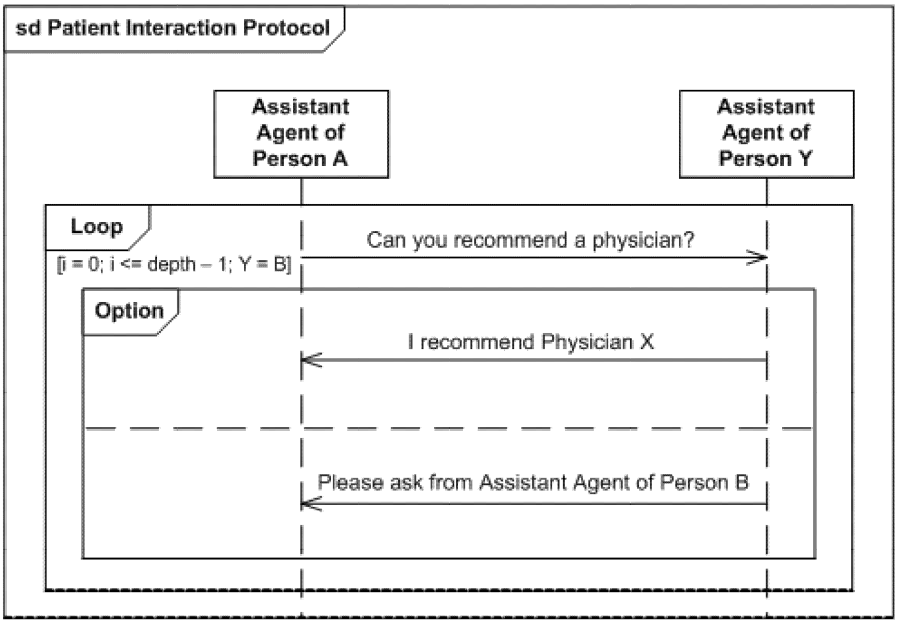

Fig. 12. The interaction protocol between patients' Assistant Agents.

Table 14
The behavioral scenario for a Human Agent playing the role of Patient

BEHAVIORAL SCENARIO 1						
Role			Patient			
Agent type			Human Agent			
DESCRIPTION						
Trigger	Condition	Step	Activity	Other roles/agent types involved	Domain entities	Relevant goals (quality goals)
A proposal by the Assistant Agent	Alternative	1	Confirm the physician	Assistant/Assistant Agent	Recommendation	Find healthcare provider (Appropriate, Good quality care)
		2	Reject the physician			
Patient enters the physician's office	Sequential	3	Receive care	Physician/Human Agent		Receive care (Discrete)
Reminder by the Assistant Agent		4	Evaluate	Assistant/Assistant Agent	Evaluation	Evaluate (In the context, Processable, Anonymous, Easy)

Table 15
The behavioral scenario for an Assistant Agent playing the role of Assistant

BEHAVIORAL SCENARIO 2						
Role			Assistant			
Agent type			Assistant Agent			
DESCRIPTION						
Trigger	Condition	Step	Activity	Other roles/agent types involved	Domain entities	Relevant goals (quality goals)
Request by the patient	Sequential	1	Find a physician	Patient/Human Agent, Assistant/ Assistant Agent	Recommendation	Find healthcare provider (Appropriate, Good quality care)
Patient leaves the physician's office		2	Evaluate	Patient/Human Agent	Efficiency, Evaluation	Evaluate (In the context, Processable, Anonymous, Easy)

Table 16
The behavioral scenario for a Human Agent playing the role of Physician

BEHAVIORAL SCENARIO 3						
Role			Physician			
Agent type			Human Agent			
DESCRIPTION						
Trigger	Condition	Step	Activity	Other roles/agent types involved	Domain entities	Relevant goals (quality goals)
Patient enters the physician's office		1	Give care	Patient/Human Agent	Capacity, Efficiency	Receive care (Discrete)

5.2. Developing a simulation for healthcare

In Section 5.1 we described how a real societal information system of healthcare should be designed. Just like the case study of grocery shopping, we simulated the healthcare case study on the NetLogo environment. In order to implement the agent-oriented models on NetLogo, we map roles of a socio-technical system that are normally performed by human agents, such as Patient and Physician in the example of healthcare, to software agent types. In addition, we make the following assumptions for the healthcare case study:

- We only address physician office visits, that is, we only consider the Physician part of the organization model represented in Fig. 9.
- We do not distinguish between diseases.
- In accordance with the quality goal "Quickly" introduced by the goal model shown in Fig. 8, we assume that a patient is willing to get healthy as soon as possible.

We next describe from three viewpoints how we mapped agent-oriented models of the societal information system of healthcare to the programming constructs of NetLogo.

From the viewpoint of *platform-dependent information design*, we represented in the simulations the domain entities introduced by agent-oriented modeling in Section 5.1 as the following NetLogo variables:

- The Capacity domain entity – in terms of the number of patients per day that a given physician can handle.
- The Efficiency domain entity – in terms of the number of days that it takes for a given physician to cure a patient. This number of days is generated for each physician according to the

normal distribution whose mean and standard deviation can be adjusted in the user interface.
- The Evaluation domain entity – in terms of the following variables:
  - The number of days the physician in question failed to handle a given patient. How this value is determined is explained below.
  - The number of days that the physician required to cure a patient. This is determined by the Efficiency knowledge item pertaining to the physician.
  - A random component representing that different patients evaluate the same physician differently.

A patient's evaluation for a specific physician is calculated by adding these three factors. For example, let us assume that a patient gets sick today and wants to go to a chosen physician, but the physician is busy and cannot see the patient until tomorrow. In this case, the value of the first factor is 1 because the patient waits for 1 day to see the physician. The second factor – number of days that the physician requires to cure the patient – is a fixed number only related to the physician. The third factor is a random number that varies from –0.5 to 0.5.

The viewpoint of *platform-dependent behavior design* covers the behaviors of software agents representing patients and physicians. In accordance with the behavioral scenarios represented in Tables 14–16, every day the patients each try to decide which physician to visit. For each patient, at step 1 of the behavioral scenario represented in Table 15, the Assistant Agent acting on behalf of its principal may ask Assistant Agents of the principal's friends for recommendations and then makes a decision as to which physician the principal should visit.

From the viewpoint of *platform-dependent interaction design*, the exchange of messages to be implemented is modeled as an interaction diagram in Fig. 12. According to the interaction diagram, the Assistant Agent acting on behalf of the patient's friend may deal with the request in one of the following ways:

- Reply with a recommendation.
- Provide the requesting agent with the address of the Assistant Agent of one of its principal's friends if there is no recommendation to give. This process continues recursively until the first recommendation is received or until all the friends until the maximum forwarding depth have been asked. The forwarding depth is defined as follows: the originator's friends are at depth 1; the originator's friends' friends at depth 2, and so on.

If the recommended physician does not have capacity on the given day, the Assistant Agent will initiate a new round of requests modeled in Fig. 12. The process continues until a patient finds an available physician. To make the simulations more realistic, in the simulation we have chosen a 20% probability that a friend would ignore the patient's request.

Returning to the *viewpoint of platform-dependent behavior design*, the software agent corresponding to the Assistant Agent recommends physicians based on evaluations. The agent can recommend only those physicians that its principal has actually visited in the simulation. The number of days the physician in question could not handle the given patient, because of the physician's exceeded capacity, accumulates in the patient's evaluation until the patient actually visits the given physician. On each new visit the agent "forgets" its previous evaluation and updates its knowledge base with the new evaluation. The reason why the agent forgets its previous evaluation is that during the time period between the previous evaluation and the new evaluation, factors that influence the evaluation may have occurred. For example, the physician may have become more skilled. Therefore it is fairer to use the latest evaluation.

To make our simulations as realistic as possible, we used the following statistical data by the Centers for Disease Control and Prevention (CDC) from the year 2008 [3]:

- The number of physician office visits per 100 people per year: 320.1.
- The number of physicians per 10,000 people: 26.

Based on the above data, we obtained the average number of people who get sick every day by dividing the number of visits per 10,000 people by 250, which is the standard number of working days in a calendar year in the U.S. As a result, 128 people in our simulation get sick every day.

We simulated 182 days with 10,000 patients. The value of the local variable of each physician's software agent corresponding to the Capacity domain entity was set to 8 patients per day. The value of the local variable of each physician's software agent corresponding to the Efficiency domain entity was determined randomly according to the normal distribution with the value of mean as 3 days and with the value of deviation as 2.0.

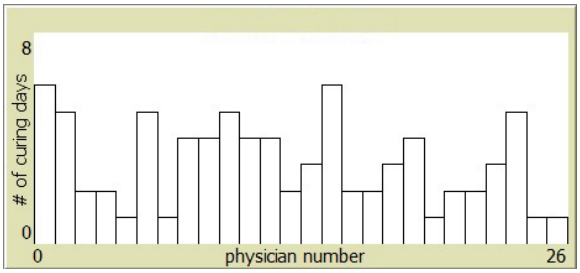

Fig. 13. The number of days to be cured by different physicians.

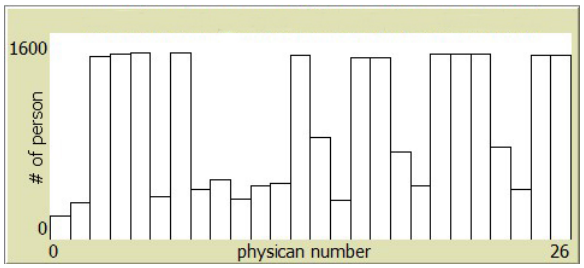

Fig. 14. The number of people visiting different physicians in the simulation.

Figure 13 shows the number of days different physicians in the simulation required to cure a patient. Figure 14 shows the number of people visiting different physicians in the 182 days. According to Figs 13 and 14, whenever the number of days required by a physician to cure a patient is small, the given physician has more patients in total. Due to space limitations, we did not include the information of how the number of people visiting different physicians changed in time. But according to this information, we can conclude that as time passes, people gradually gather information about physicians, evaluate them, and recommend to their friends the best physicians they know. As a result, after patients have formed their opinions about the physicians, high quality physicians get full capacity of patients every day and low quality physicians get nearly zero patients. This is quite similar to real life, because people always prefer good physicians.

6. Related work

Information systems for controlling the distribution of electric power among individual consumers have been investigated in a series of articles from Jennings and colleagues [8,35]. The investigations are the closest to the research presented here. The information systems assist consumers in dealing with the complexities of a global electric power distribution system while respecting their individual preferences. However, unlike the systems we have analyzed here, there is no interaction among the consumers for decision-making or control: a consumer's decision affects the other consumers, but is made in isolation. Such systems help individuals in reserving and using societal resources, but they do not help the individuals in acting collectively or collaboratively. For example, the system described in [8] makes use of agents that compete via an auction. As a result, the set of individual consumers is not an equal partner with the purveyor of the resources, such as a grocery store, hospital, or electric power company. As a result, consumers are at a disadvantage.

We are not the first authors who explore the principles of societal shopping. Price comparison services (also known as comparison shopping services) allow people to query the prices of a product at online stores. The services list the product's prices in all of the stores and sort the prices to provide customers with support for their online shopping. An intelligent software agent to implement comparison shopping is called a shopbot [5]. Shopbots have also given the name for the type of agents assisting customers in our case study of societal grocery shopping.

In June 1995, the first well-known shopbot called BargainFinder [13] was released as an intelligent software agent for comparison shopping for music CDs. It allowed a user to enter the name of an artist and an album, searched eight online music stores, and displayed all CD prices on a webpage. If the user clicked on the name of one of the stores, it would bring the user to the specific album on that store's website. Customers gained obvious benefit from BargainFinder and it has been used widely. Nowadays, shopbots have greater functionality than before by including information about shipping expenses, taxes, vendors' rates, and product reviews. Some corporations even have their own shopbots [9,26]. Similarly to shopbots, the iPhone application RedLaser [24] accepts the barcode of a product from the phone's camera, searches many online stores, and shows their prices on the phone. It, however, relies only on the information it retrieves from online stores and does not help its user in deciding stores and shopping baskets.

Regarding the case study of finding healthcare providers, Udupi & Singh [31] emphasize the importance of conceptual models in developing societal information systems. They claim that the conceptual model should support social knowledge as cleanly

separated from domain knowledge. They also claim in [31] that healthcare is a natural fit for peer-to-peer (P2P) service networks and describe a scenario where a patient has as neighbors his primary care physician and his close friends, and contacts them to request services or referrals. The emphasis of their work is on the adaptation of a social network, while we focus on the effectiveness of finding a good quality medical service provider.

In [17] the Personal Health Server developed in Finland is described. It is a system designed to assist healthcare workers, patients, and their families with medical information and services, and help them make appropriate decisions. The Personal Health Server will enable disparate e-health tools to work together and share a computer glossary of terms, definitions, and their relationships. The difference from our approach is that it is a centralized client-server system while our approach is a distributed P2P solution.

There are some websites, like RateMDs [23], where people can rate and find physicians. The system proposed by us differs from RateMDs and other similar websites in the way people rate the physicians and in the way patients interact. Such websites use criteria like punctuality, medical knowledge, and time spent on a patient, while we use the time it took to be cured, which is a more objective criterion. Although a patient may access more ratings online, he usually does not know the people who have rated the physicians. In our system, a patient relies on friends' recommendations, which are more reliable.

7. Discussion and conclusions

Wang, Zeng, Carley, and Mao [36] emphasize that communities are increasingly driving innovation from the bottom up, and the ownership of experience, economic value, and authority is starting to shift from institutions to communities. According to Forrester Research Report [25], individuals in today's world have more power than ever before because mobile, social, video, and cloud technologies give individuals tremendous access to information and resources. However, because of the huge amount of information available, the effort required from individuals to influence the state-of-affairs is also substantial. To make the effort reasonable and manageable, we propose in this article to represent each individual – be it a customer in grocery shopping or a patient in healthcare – by a software agent that acts in the interests of its principal and interacts with the agents of other individuals.

For designing information systems that are aimed at supporting individuals acting within social networks, both social and technical aspects of such information systems should be considered. We have chosen to use agent-oriented modeling for designing societal information systems because this approach explicitly addresses the engineering of socio-technical systems where the activities of humans are supported by software agents. What makes agent-oriented modeling particularly appropriate for designing societal information systems is that the engineering process starts with specifying goals for a socio-technical system as a whole and defining roles required for achieving the goals. Technical and social subsystems of the system are identified only later in the design process when roles are mapped to the types of agents enacting them. That is also a stage when the decisions of architectural design can be made by mapping roles to different possible configurations of agents. Alternatively, the system architecture can be designed already when deciding roles. For example, in the case study of grocery shopping, we introduced the role Coordinator already at the beginning of analysis. Later on at the design stage this role was straightforwardly mapped to the agent type Coordinator Agent.

When designing an information system for a problem domain, agent-oriented modeling enables addressing the problem domain from three balanced perspectives: information, interaction, and behavior. In our case studies we covered all three perspectives for the analysis, as well as for platform-independent design and platform-specific design for NetLogo.

Another advantage of agent-oriented modeling is that its behavioral scenarios do not presume any particular agent architecture, such as BDI [22]. This means that in agent-oriented modeling, deciding the agent architecture(s) is postponed to the stage of platform-dependent design, which is preferable for two reasons. First, since all agents of the system do not need to be designed in the same way, one can find the most appropriate architecture for each software agent type of the system to be designed. Second, at the stage of platform-dependent design, the platforms and technologies available for implementing the system are determined, which may constitute additional constraints on deciding the agent architecture(s). Moreover, some agents, such as the Coordinator Agent in Section 4.1, may be implemented as a web service running in a cloud rather than software agents.

In the future, we plan to implement the societal information systems of grocery shopping and healthcare on a platform for multi-agent systems with real rather than simulated interactions between software agents. The resulting systems will then be tested in real-life case studies involving college students.

As this work did not address the aspect of how the social network forms and evolves, we plan to include this as an important part of our future work.

References

[1] A. Berson, *Client/Server Architecture*, 2nd edn, McGraw-Hill, 1996.

[2] P. Bresciani, A. Perini, P. Giorgini, F. Giunchiglia and J. Mylopoulos, Tropos: An agent-oriented software development methodology, *Autonomous Agents and Multiagent Systems* **8**(3) (2004), 203–236, Springer-Verlag.

[3] Centers for Disease Control and Prevention, Ambulatory Care and Physician Visits, http://www.cdc.gov/nchs/fastats/docvisit.htm, retrieved: 19th June 2011.

[4] L. Cernuzzi, T. Juan, L. Sterling and F. Zambonelli, The GAIA methodology: Basic concepts and extensions, in: *Methodologies and Software Engineering for Agent Systems: The Agent-Oriented Software Engineering Handbook*, F. Bergenti, M.-P. Gleizes and F. Zambonelli, eds, Kluwer Publishing, 2004, pp. 69–88.

[5] D. Clark, Shopbots become agents for business change, *IEEE Computer* **33**(2) (2000), 18–21, IEEE Computer Society.

[6] S.A. DeLoach and M. Kumar, Multiagent systems engineering: An overview and case study, in: *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini, eds, Idea Group, 2005, pp. 317–340.

[7] H. Du and M.N. Huhns, A Multiagent system approach to grocery shopping, in: *Advances on Practical Applications of Agents and Multiagent Systems - 9th International Conference on Practical Applications of Agents and Multiagent Systems (PAAMS 2011)*, Salamanca, Spain, 6–8 April 2011, Y. Demazeau, M. Pechoucek, J.M. Corchado and J.B. Perez, eds, Advances in Intelligent and Soft Computing, Vol. 88, Springer-Verlag, 2011, pp. 195–200.

[8] E. Gerding, V. Robu, S. Stein, D. Parkes, A. Rogers and N. Jennings, Online mechanism design for electric vehicle charging, in: *The 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, May 2–6, 2011, Taipei, Taiwan, Proc., P. Yolum, K. Tumer, P. Stone and L. Sonenberg, eds, International Foundation for Autonomous Agents and Multiagent Systems, 2011, pp. 811–818.

[9] Google, Google Product Search, http://www.google.com/products/, retrieved: 19th June 2011.

[10] M.N. Huhns, From DPS to MAS to …: Continuing the trends, in: *The 8th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2009)*, May 10–15, 2009, Budapest, Hungary, Vol. 1, C. Sierra, C. Castelfranchi, K.S. Decker and J.S. Sichman, eds, ACM, 2009, pp. 43–48.

[11] M.N. Huhns and L.M. Stephens, Multiagent systems and societies of agents, in: *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*, G. Weiss, ed., MIT Press, 1999.

[12] T. Juan, A.R. Pearce and L. Sterling, ROADMAP: Extending the Gaia methodology for complex open systems, in: *The First International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2002)*, July 15–19, 2002, Bologna, Italy, Proc., ACM, 2002, pp. 3–10.

[13] B. Krulwich, The BargainFinder agent: Comparison price shopping on the Internet, in: *Agents, Bots and Other Internet Beasties*, J. Williams, ed., SAMS.NET, 1996, pp. 257–263.

[14] Y. Luo, L. Sterling and K. Taveter, Modelling a smart music player with a hybrid agent-oriented methodology, in: *Proc. of the 15th IEEE International Requirements Engineering Conference*, October 15–19, 2007, Delhi, India, IEEE Computer Society, 2007, pp. 281–286.

[15] A. McAfee, Enterprise 2.0: The dawn of emergent collaboration, *MIT Sloan Management Review* **47**(3) (2006), 21–28, MIT Press.

[16] M. Mecella, M. Angelaccio, A. Krek, T. Catarci, B. Buttarazzi and S. Dustdar, WORKPAD: An adaptive peer-to-peer software infrastructure for supporting collaborative work of human operators in emergency/disaster scenarios, in: *International Symposium on Collaborative Technologies and Systems (CTS 2006)*, IEEE Computer Society, 2006, pp. 173–180.

[17] News-Medical.Net, Computer-Based Tools Can Help Improve Relationship Between Patients, Healthcare Workers, http://www.news-medical.net/news/20110317/Computer-based-tools-can-help-improve-relationship-between-patients-healthcare-workers.aspx, retrieved: 20th June 2011.

[18] T. Nguyen, S.W. Loke, T. Torabi and H. Lu, PlaceComm: A framework for context-aware applications in place-based virtual communities, *Journal of Ambient Intelligence and Smart Environments* **1**(3) (2011), 51–64, IOS Press.

[19] Object Management Group, Unified Modeling Language: Superstructure, Version 2.1.1, February 2007, http://www.omg.org/cgi-bin/doc?formal/07-02-05, retrieved: 28 June 2007.

[20] J. Paay, L. Sterling, F. Vetere, S. Howard and A. Boettcher, Engineering the social: The role of shared artifacts, *Int. J. Human-Computer Studies* **67**(5) (2009), 437–454, Academic Press.

[21] L. Padgham and M. Winikoff, *Developing Intelligent Agent Systems: A Practical Guide*, John Wiley and Sons, 2004.

[22] A.S. Rao and M.P. Georgeff, Modeling rational agents within a BDI architecture, in: *Proc. of Knowledge Representation 91 (KR-91)*, J. Allen, R. Fikes and E. Sandewall, eds, Morgan Kaufmann, 1991, pp. 473–484.

[23] RateMDs, Find and Rate Doctors and Dentists, http://www.ratemds.com/, retrieved: 15th January 2012.

[24] RedLaser, http://www.redlaser.com/, retrieved 19th June 2011.

[25] T. Schadler, S. Leaver, J. Bernoff and A. Yakkundi, An Empowered Report: Reinvent Yourself yo Serve Empowered Customers And Employees, http://www.forrester.com/rb/Research/welcome_to_empowered_era/q/id/57265/t/2, retrieved 22nd April 2011, Forrester Research, 2011.

[26] Shopping.com, http://www.shopping.com/, retrieved: 19th June 2011.

[27] I. Sommerville, *Software Engineering*, 8th edn, Addison-Wesley, 2007.

[28] L. Sterling and K. Taveter, The Art of Agent-Oriented Modeling, MIT Press, 2009.

[29] L. Sterling, T. Miller, K. Taveter, B. Lu and G. Beydoun, Requirements Engineering Using the Agent Paradigm: A Case Study of an Aircraft Turnaround Simulator, Working Article, http://ww2.cs.mu.oz.au/~tmill/pubs/aore.pdf, retrieved: 25th January 2012.

[30] K. Taveter and G. Wagner, Towards radical agent-oriented software engineering processes based on AOR modeling, in: *Agent-Oriented Methodologies*, B. Henderson-Sellers and P. Giorgini, eds, Idea Group, 2005, pp. 277–316.

[31] Y.B. Udupi and M.P. Singh, Information Sharing among Autonomous Agents in Referral Networks, in: *Agents and Peer-to-Peer Computing, 6th International Workshop (AP2PC 2007)*, May 14–18, 2007, Honululu, Hawaii, USA, Revised and Selected Articles, S.R. Joseph, Z. Despotovic, G. Moro and S. Bergamaschi, eds, Lecture Notes in Computer Science (LNCS), Vol. 5319, Springer-Verlag, 2007, pp. 13–26.

[32] United States Department of Labor, Consumer Price Index, http://www.bls.gov/cpi/#tables, retrieved: 19th June 2011.

[33] F. Vetere, H. Davis, M. Gibbs and S. Howard, The magic box and collage: Responding to the challenge of distributed intergenerational play, *Int. J. Human-Computer Studies* **67**(2) (2009), 165–178, Academic Press.

[34] F. Vetere, M.R. Gibbs, J. Kjeldskov, S. Howard, F. Mueller, S. Pedell, K. Mecoles and M. Bunyan, Mediating intimacy: Designing technologies to support strong-tie relationships, in: *Proc. of the 2005 Conference on Human Factors in Computing Systems (CHI 2005)*, April 2–7, 2005, Portland, OR, G.C. van der Veer and C. Gale, eds, ACM, 2005, pp. 471–480.

[35] P. Vytelingum, T.D. Voice, S.D. Ramchurn, A. Rogers and N.R. Jennings, Agent-based micro-storage management for the smart grid, in: *The Ninth International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, May 10–14, 2010, Toronto, Canada, Proc., ACM, 2010, pp. 39–46.

[36] F.-Y. Wang, D. Zeng, K.M. Carley and W. Mao, Social computing: From social informatics to social intelligence, *IEEE Intelligent Systems* **22**(2) (2007), 79–83, IEEE Computer Society.

[37] D. Weyns, A. Omicini and J. Odell, Environment as a first class abstraction in multiagent systems, *Autonomous Agents and Multiagent Systems* **14**(1) (2007), 5–30, Springer-Verlag.

[38] U. Wilensky, NetLogo, http://ccl.northwestern.edu/netlogo/, retrieved: 22nd April 2011, Center for Connected Learning and Computer-Based Modeling, Northwestern University, 1999.

[39] D. Wilmann and L. Sterling, Guiding agent-oriented requirements elicitation: HOMER, in: *The 2005 NASA/DoD Conference on Evolvable Hardware (EH 2005)*, 29 June–1 July, 2005, Washington, DC, USA, IEEE Computer Society, 2005, pp. 419–424.

[40] M. Wooldridge, *An Introduction to Multiagent Systems*, 2nd edn, John Wiley and Sons, 2009.

[41] F. Zambonelli, N.R. Jennings and M. Wooldridge, Organizational abstractions for the analysis and design of multiagent systems, in: *Agent-Oriented Software Engineering, First International Workshop (AOSE 2000)*, Limerick, Ireland, June 10, 2000, Revised Articles, P. Ciancarini and M. Wooldridge eds, Lecture Notes in Computer Science (LNCS), Vol. 1957, Springer-Verlag, 2001, pp. 235–251.