Decision Making with a KGP Agent System

Jeremy Forth* — Kostas Stathis — Francesca Toni***

**Department of Computing*
Imperial College London

jforth@doc.ic.ac.uk
ft@doc.ic.ac.uk

***School of Informatics*
City University London

kostas@soi.city.ac.uk

ABSTRACT. *A human user is centrally involved in a Decision Support System (DSS) task, whereas for an autonomous agent system, a user's role is a more abstract notion of oversight and approval. In this paper, we examine the potential integration of a multi-agent architecture into a DSS. The motivation in this work is to utilise autonomous agent problem-solvers within a suitable multi-agent framework in such a way as to realise and generalise the capabilities of a DSS, while maintaining the DSS characteristic of significant user involvement. The approach taken is to employ agents both as domain problem solvers as well as to manage the interaction with the user as part of the semi-autonomous design. We consider the construction of an agent-based DSS based upon the KGP model of agency and detail how the design of a DSS may benefit from agent technology in general and KGP agents in particular (and vice versa). A framework for agent-enhanced DSS is presented which is more general than those presented previously. We show how the architectural framework used for KGP agents can be specialised to realise enhanced DSS capabilities.*

RÉSUMÉ. *L'ensemble des.*

KEYWORDS: *Logic-based Agents, Decision Support Systems, KGP Model, Multi-agent Systems.*

MOTS-CLÉS: *un maximum de six mots significatifs doivent être isolés sous forme de mots-clés.*

1. Agents in DSS - Introduction

Decision Support Systems (DSS) were originally formulated in works such as those of (Morton 1971), (Keen *et al.* 1978), (Bonczek *et al.* 1980) and (Sprague *et al.* 1982). In a typical role, the systems aid humans to make informed decisions for problem-solving over a targeted domain by aggregating information from that domain and performing data analysis according to various predefined numerical models. Through simulations based on these models, projections may be made under a range of hypothetical scenarios using varying assumptions.

A classical DSS system is comprised of a Dialog Generation and Management System (DGMS), Database Management System (DMS), and a Model Base Management System (MBMS). The paradigm of use is typically that of direct manipulation of a passive system by a user. This system provides information in support of a decision about an appropriate course of action to a user who will assimilate the information and apply it in conjunction with their worldly expertise to render a good result. The user is also expected to execute the chosen course of action.

There is substantial variation to be found in the literature in the use of the term "Decision Support System". It may be used to refer alternatively to a classical DSS as described above, or more generally to a group of information systems with a capacity to support human decision making. In this latter more general use (Power 2002) has classified five different types of DSS: model driven (e.g. financial models), communications driven (e.g. collaboration tools), data driven (e.g. daily sales reports), document driven (e.g. digital library), and knowledge driven (e.g. expert system).

General artificial intelligence capability has been recognised as valuable to classical DSS, see for example (Radermacher 1994). A classical DSS may accordingly be extended by the addition of logical reasoning capability such as that found in an Expert System (Ignizio 1991). Domain-specific declarative knowledge may be encoded in such a way that the raw output of the numerical models are subjected to suitable filtering, interpretation and analysis before final considered advice with supporting information is provided to the user. The user is no longer required to possess all the domain expertise necessary for problem resolution. DSS and Expert Systems thus integrate together very powerfully to automate a greater portion of the full decision-making process, thereby reducing the demands that a complex conventional DSS places on the user's knowledge (Turban *et al.* 2006).

One of the shortcomings of Expert Systems was that disparate expertise contained within individual systems' knowledge bases could not be shared with one another. A more recent, and more powerful paradigm currently prevalent in AI research, the autonomous agent and multi-agent system (MAS), successfully addressed some of these communicative shortcomings of Expert Systems. Possibly as a result, MAS requires more consideration for suitable integration with DSS, however it arguably offers more opportunity, see (Angehrm 1993) and (Mora *et al.* 2003). Agents are designed to be autonomous problem-solvers, possibly communicating with other agents and users, and are therefore equipped with sufficient cognitive abilities to reason about a domain, make certain types of decisions themselves, and perform the associated actions. They offer the potential to automate a far wider part of the overall problem-solving task than was possible with classical DSS or Expert System DSS. However, since agents can only approximate the breadth or depth of expertise that humans bring to the activities of decision making and problem-solving, human participation will always likely yield superior results.

An autonomous agent setting usually considers the agent situated in an environment over which it performs direct sensing and perception, and also exercises direct control through the use of actions for the purposes of initiating change. Although collaboration with other agents is a central feature of multi-agent systems, the focus of design is on autonomy of the system, not particularly on collaborating with human users. A typical example is the Retsina (Sycara *et al.* 1996) framework's user interface agent, which receives a goal and task specification from the user and presents results in return. This type of user involvement offers little flexibility for maintaining a sufficiently rich dialog with the user. In traditional agent system designs, the agent makes most decisions by itself, in contrast to a classical DSS design where the human user drives the decision support tool (through direct control of the system) and performs all decision-making. This is perhaps the most pertinent difference between a MAS and a DSS.

One can view the DSS setting and agent setting as being on opposite extremes of a continuum of collaborative decision-making between the system and its users: representing direct manipulation versus oversight and approval respectively. The contribution made to agent system design through consideration of a DSS setting is therefore the introduction of a notion of rich bi-directional collaboration with the user about the circumstances and choices arising during task execution. Agent systems in turn contribute to DSS by providing autonomous problem-solving capability and interaction with other agents. We believe a system designed to fit midway in this continuum contains the most power for problem-solving in organisational settings. The agent system and user are then assumed to work together in collaborative problem-solving, each aware and appropriately responsive to the other.

Incorporation of agents also offers the opportunity for the system to participate directly in the implementation of decisions through the execution of actions directly upon the domain. The *planning* capability of agents may directly generate multiple decision-making alternatives for the user which are based on alternative specific courses of action. An agent-enhanced knowledge-based DSS can therefore potentially provide a more direct level of decision support when compared with more classical DSS designs based on numerical analysis information, other indirect indicators, and perhaps Expert Systems. A plan generated by an agent as a potential decision alternative may describe physical actions to take, or alternatively describe an organisational process. These two plan types differ in the level of abstraction of actions. Physical actions are direct-acting on the domain, while organisational planning actions usually require the construction of conditional plans (more akin to a flowchart), and may be formulated from a plan library for standard subtasks. An agent-based DSS may be used to achieve a specified outcome, maintain goals, detect deviation conditions from these goals, generate restoration alternatives, and direct pursuit of the remedy.

In this paper, we examine the potential integration of agent technology into a framework of a semi-autonomous DSS. This is done in a way which preserves a human user's potential to contribute their high level skills, while still benefiting from a high degree of automation in decision-making and the wider problem-solving task. The paper explores some of the proposals presented in (Stathis *et al.* 2005) to promote the views to a wider audience including DSS. In section 2 we present a standard decision process. Section 3 defines how the capability of an agent may be utilised in a DSS application, and also identifies alternative agent design architectures suitable to underpin this. Section 4 describes the KGP agent and shows how this particular agent design is used to realise DSS functionality. Specific examples showing how this is achieved are given. Section 5 considers the properties of decision making as realised in the KGP-DSS and considers the resulting overall decision-making capability in comparison to other decision-making methodologies found in the literature. Section 6 identifies interesting related work, and section 7 concludes.

2. Decision Making

Human decision making on behalf of organisations is distinct from decision making on behalf of one's self. Humans use their own decision making capability to facilitate a shared organisational decision making process, and in the same way so autonomous agents may apply their own autonomous decision making capability to assist in a shared decision making process in which humans participate.

Shared decision making is typically a complex multi-stage process. There have been many attempts to formalise the procedure, and it has been frequently noted, e.g. (Mankins *et al.* 2006), that procedures of any kind are often not followed in practice. For the purposes of analysing agent contributions to a decision making process, it is important that we classify at least one process. For this purpose, we will choose Simon's model (Simon 1960) which is a very well accepted top level description of decision making processes.

The model specifies the following main abstract stages of decision making: intelligence, design, and choice. Also implicit in Simon's decision stages is the final stage of decision implementation. Table 1 shows this process arrangement with each stage broken down into sub-tasks in a way compatible with those suggested in

(Turban *et al*. 2006). Using this decision process, we will now proceed to discuss the ways in which autonomous agents may be integrated with decision support system design.

Decision Phase	Sub Tasks
Intelligence	- Goal formulation. - Data collection. - Problem decomposition. - Approval and ownership. - Problem statement and framing.
Design	- Model selection. - Formulating possible courses of action. - Select suitable principles of choice.
Choice	- Evaluate alternatives based on principles of choice. - Selection of the highest ranking alternative.
Implementation	- Carry out the course of action in the environment.

Table 1. *Decision making based on Simon's model.*

3. Alternative Agent-Based DSS Designs

3.1. *Detailed Functionality*

The overall cognitive capability present in an agent can be deployed in several ways to realise effective problem-solving by the combined effort of the system and user. A user may typically have high-level expertise about a situation while a system may have detailed low-level domain knowledge. In typical use, a DSS supplies information in support of a decision process performed by a user, while in agent systems by contrast, the human user typically expresses a domain specific goal to the agents without the opportunity of imparting all their high level expertise. Therefore, in agent systems there is little opportunity for the system to benefit from the user's domain expertise, and insubstantial feedback pathway for the user to be educated about any detailed specific difficulties encountered by the agent in pursuing a particular solution.

This disconnect in the user's feedback path in a standard agent system is solved in the DSS setting where the system explains its decisions in a form useful to a high-level-reasoning user. Agent resources must be specifically allocated to solve this "user management" task.

Knowledge types represented by agent systems are also different to those from a standard DSS setting. There is a environmental causality-based domain description and there is a description of possible information sources. There is also a knowledge base containing information about how users should be treated in general, and particular knowledge about individual users. These descriptions may contain forms of weaker knowledge, which is important for its part in the formation of conclusions in the presence of incomplete information (where no conclusions could otherwise be drawn), or to arbitrate among inconsistent conclusions. Weak knowledge may be expressed as preference information for example. This may be used to model the domain, or the user, and is one possible basis from which to support autonomous decision-making.

3.1.1. *Roles of a DSS Agent*

When developing an agent based DSS, there is a choice over which of two possible kinds of actors, the user or domain agent, will execute the chosen course of action, and which type of actions the agents will perform in the environment. An agent, for instance, cannot go to the theatre, but it could book a ticket. Clearly, information acquisition actions in pursuit of knowledge goals, and communicative actions in reaction to environmental changes are less invasive than direct-effect causation actions in pursuit of the user's goals. The greater the extent of an agent's role in the execution of the solution, the further the design departs from the classical DSS setting (as outlined previously) and the more general the design becomes.

DSS Function	**Agent Function**
Data collection	Knowledge acquisition and assimilation.
Model creation	Perception and knowledge representation.
Alternative case creation	Planning and reactivity.
Choice	Action selection.
Implementation	Action execution.

Table 2. *Mapping DSS functions to Agent capabilities.*

Table 2 shows one particular way functions of a DSS may be realised as an agent based design. As a constituent part of problem-solving in the domain, an agent may choose particular sources of information to use. Data Gathering may be a function within an agent (sensing), or a dedicated activity of a specialised information agent if the task is complex. Model creation and maintenance is performed continuously by an agent by a combination of its precepts and knowledge assimilation through belief revision. The goal is to always maintain an accurate model of the domain for the purposes of the class of tasks to be performed. Creating alternative courses of action is part of agent problem-solving realised by planning and reactivity. Agents reason about goals, sub-goals, and actions that realise those goals. Once created, these alternatives can be arbitrated among either by the agent itself, or alternatively through communication with another agent or with the user who then makes the final selection choice.

3.1.2. *User Management and Domain Roles*

Communication with the user is maintained by an agent fulfilling the role of User Management. We call this agent the User Management Agent (UMA) because its objectives are to manage how the user understands the system's activity and how the system understands the user. Consequently, this role is a superset of the more usual interface agent found in agent system designs. Communication between the two actors (user and system), pursuing collaborative problem-solving is possible with respect to plans of action (generated by the system or user), and factual statements, in addition to the more usual model-based analysis. Typically the user manager would perform communication with the user about alternative possible courses of action and their known consequences, seek approval for courses of action over which the user may be sensitive, and accept or clarify user directives. This may occur as a reaction to opportunities and risks occurring in the domain, or be driven by proactive requests made to the user. As a communicating device, the user manager may also facilitate contact between various interested parties. Example goals for the user management agent are:

– user is aware of plausible alternatives;

– user understands the fundamental domain structure;

– user understands consequences of following a selected course of action;

– user has been notified and has accepted the execution of actions known by the system to have wide ranging effects, or to be sensitive;

– user has been informed of unexpected opportunities or problems encountered.

The Domain Agent (DA) situated problem solver is ideally placed to generate new options based on its declarative knowledge base, optionally augmented with numerical models. Using this understanding of the domain, it can enumerate alternative courses of action along with relevant known consequences, whether the choices originated with the user or another agent. Justification for a choice is given by detailing the objectives (goals), pertinent to the user, potentially conflicting consequences, and general information in support of the choice. In this way, the user is informed about what is at issue only if it is likely to be of importance.

Proposals are communicated and justified on the basis of decision-making occurring about the user on the basis of a user model. The system is expected to maintain a user model, and this may be (partly) realised as a user preference model. This model determines what to bring to the user’s attention, so that the user is solely informed by relevant and important facts. The system is thus engaged in an act of “managing up” to the user. This entails the agent taking active responsibility for the user’s proper involvement. The effectors of user management agents are user interface modifiers in the form of communication mechanisms for information presentation. These may manage multiple users though broadcasting, and individuals through dialogs. We assume that agent-user interaction dialogs conform to well-specified protocols (Pitt *et al.* 1999) like those specified by standards bodies such as (FIPA 2002).

These protocols are instantiated to be suitable for conducting a conversation for guiding a decision-making process. They will accordingly define the decision-making purpose of each exchange. This will include provision for a user to state a “goal” for which a decision over a plan of action is required, and identify the potential actions that are the subject of the conversation. A user may “request” potential alternatives, ask the agent to “justify” a solution, modify a single component part of a candidate solution: “use alternative”, or “accept plan” of action for partial or fully automated implementation. We further assume that the conversation will refer to a domain specific ontology defining the common terms used in the application domain (e.g. a user’s goal might refer to a book whose properties include an author, a title, and ISBN number, all of which must be provided for in the ontology).

The problem domain will also contain its own constraints, for example, a book may not have the same ISBN number as a different book. Solutions respecting these constraints must then in turn interact with, and elicit, the user’s latent preferences. It is more effective to formulate solutions in this way because the user’s preferences and wider goals can never be fully understood by the system. While the Domain control function will know about some domain constraints beforehand, it will discover more during the execution process. If considered important by the user manager, these restrictions are then communicated to the user and allowed to interact with the user’s own preferences through communication.

3.1.3. *Decision Types*

In general, classes of decision types may be differentiated by how a decision is taken, what it is about and who takes the decision. Decisions taken by an agent in the system can be differentiated according to whether they are about the domain, or about the user, and whether they are subject to oversight from the user. Oversight from the user must be specifically requested by the system. This depends in turn on decisions based on the user preference model describing the user's sensitivities to conditions occurring in the domain, and therefore when the user should be involved.

Decision	**Taken by**	**Choices**
Autonomous domain control	DA	Domain goals and actions.
Approval decision	UMA	Decision maker: (UMA, DA, user)
Alternative solution choice	UMA	Domain goals
User management decision	UMA	Status information
User decision	User	Domain Solutions

Table 3. *Decision types.*

There are five distinct types of decisions made by this agent-based DSS design as shown in table 3. The first type of decision, the *autonomous domain control* decision are those that determine the steps taken in attempts to reach a sub-goal in the domain. These are taken directly by the domain controlling agent, on behalf of themselves, and are internal and autonomous. An example instance is where a domain agent selects one information source over another.

The second type of decision, the *approval decision* makes a choice about what level of decision maker is appropriate for the choices pending. The three options are as follows. The domain agent may take the decision using its domain expertise, the user management agent may take the decision on behalf of the user, and lastly, the user themselves may take the decision. Depending on the sensitivity of the components of a solution, some parts of alternative solutions may be presented to the user for approval or selection if it is considered significant enough by the user manager. This decision, in effect, makes the choice about which decisions are made internally to the system, and which are made externally.

The third type of decision is the *alternative solution decision*, made by the user manager which takes lower importance decisions on behalf of the user on alternative solutions reported by domain agents. This is possible because the user manager knows much information about the user and their preferences. An example of this type of decision is to autonomously select an earlier train time.

The fourth type of decision, the *user management decision*, determines how the system will interact with the user. Depending on the agent's knowledge of the user's preferences and responsibilities, the agent makes choices about what to bring to the attention of the user. These are choices about how to "educate the user" about the circumstances. An example of this is alerting the user to a promotional offer. Another example is failure to achieve a desired user-goal. This condition may be reported to the user if there are no other alternatives available (see section 4.4.3).

The final type of decision is the *user decision*. These occur after alternative solutions have been presented to the user, and the user makes a selection. User decisions can also be externally affected by a continuously changing environment which may subsequently cause the user to change his mind, and as a result interrupt a currently active alternative explored by the system. An example of this is the user deciding to go to the theatre rather than the bookstore because of an advertisement brought to the attention of the user after a prior decision to buy a book.

3.2. *Agent Frameworks*

Agents and multi-agent systems may be configured in a number of different ways to realise the essential user management and domain control functions of a DSS described earlier. These roles may be held by a single agent, though more usually will be divided up among specialist multiple agents, each equipped with information about different users and different domain components. A single agent may function as a simple DSS providing it has sufficient resources to realise user management and domain control simultaneously. As pointed out in (Matsatsinis *et al.* 1999) the appropriate agent framework for a particular DSS depends on the capabilities of each agent employed in the design.

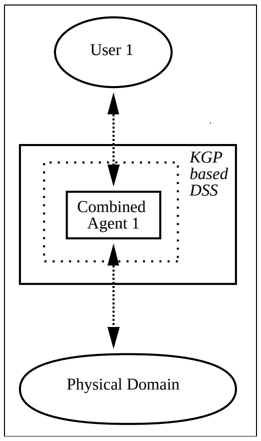

Figure 1. *Single agent DSS where user management and domain problem-solving are combined into one agent.*

In cases where the agent design is powerful enough to undertake a user management task and domain problem-solving task simultaneously, then the two functions may be combined into one agent. In this case the agent is required to play a composite role, that of the user manager and the domain problem solver (Zambonelli *et al.* 2003). The agent's internal representation distinguishes between the knowledge required for the domain expertise and the knowledge required to manage the user. Figure 1 shows a general-capability agent situated in a decision-making domain where it takes the responsibility for domain control and user management, thereby realising a simple DSS.

The usual adopted method for a MAS to interface with the user is to employ an Interface Agent as a link between DAs and the user. Its role is typically limited to receiving user instructions, involve appropriate task agents and display results (Sycara *et al.* 1996), (Laurel 1990). A DSS realisation in the form of a MAS would typically make use of a class of agents whose sole purpose is to collaborate with human users. We have chosen to introduce the notion of a UMA which takes responsibility for managing the user and the associated connection between the user and DAs. This then allows domain agents to specialise in some selected area of domain control (a particular advantage for large domains).

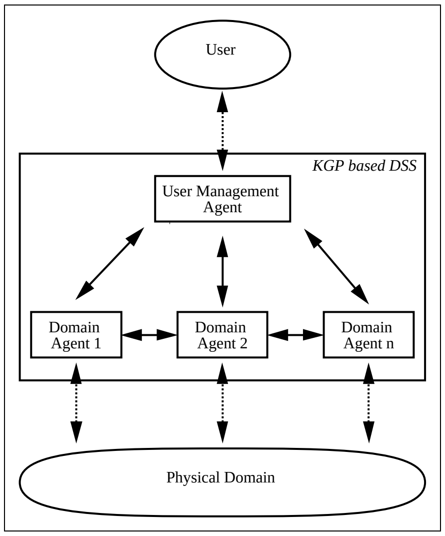

Figure 2. *Separated responsibility Single User Framework where user management and domain problem-solving are handled by different agents.*

The simplest specialised architecture realising this functionality is shown in Figure 2, where agent responsibility and expertise is partitioned according to whether an agent has knowledge and responsibility for the domain of discourse, or alternatively for the user (in terms of preferences and likely sensitivities).

Accordingly, DAs are assigned the role of controlling the domain (achieving a desired outcome), whilst the UMAs are assigned the role of managing the user. In a similar way as DAs meet a goal for the domain, the user management agent meets user management goals.

The user management agent takes responsibility for maintaining a coherence of context between the user and system. As well as the communication of simple task specifications, goals and results between the user and system, as an Interface Agent would do, the UMA implements the user management function. Its most important task therefore is collaborative conflict resolution and solution formulation. The UMA also communicates with other agents involved in problem solution. These may be divided into specific domain roles such as domain information agent, domain task agent, or domain controller, depending on the application.

Domain agents are the only agents to connect to the domain. They do so through physical actuators such as motorised machines or network interfaces for electronic domains. The limitations of Figure 2 are that it is a simple user architecture in the sense that it does not permit multiple users to collaborate in solving a problem as is typical in an organisational setting. Even if instantiated multiple times, Figure 2 only collaborates between domain agents on the problem's solution, it does not facilitate user collaboration. Depending on the capability of the underlying agents, it is possible to extend the architecture to allow a UMA representing one user to interact with another UMA representing a second user.

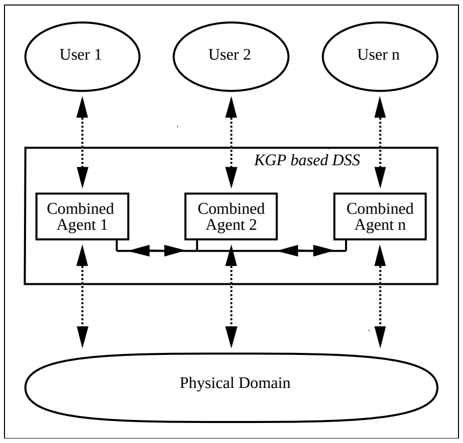

Figure 3. *Combined responsibility Single Resource Framework with multiple agents and users.*

Figure 3 introduces this enhancement. Assuming the agents are powerful enough to interleave cognitive tasks, then user management and domain controller functions can be combined. The architecture is naturally able to support multiple agents collaborating on domain tasks, while also allowing multiple users to collaborate with the system. This architecture therefore represents the minimum design requirement for use in an organisation.

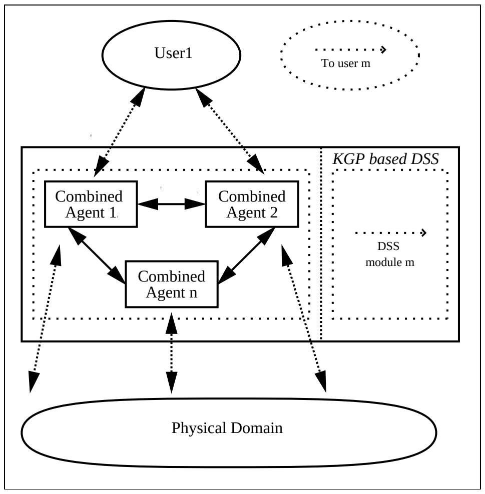

Figure 4. *Combined responsibility multi-resource framework where a user has a cluster of dedicated agents for decision-making support.*

Figure 4 is a more advanced multi-resource framework for DSS where each user is allocated a cluster of dedicated agents each specialising, and each able to interact with other agents, the environment and the user. This is suitable for larger multi-faceted problems containing many different aspects requiring multiple specialised model types and where users typically cater to more than one goal of the overall problem solution. The demands placed on each agent of Figure 4 are user management, coordination between other user management agents and coordination between other domain agents working on the same task, and others working on related but different tasks.

4. Using KGP Agents for DSS Applications

We advocate a particular kind of agent in this work, built according to the Knowledge-Goals-Plan (KGP) model of agency (Kakas *et al.* 2004). The KGP model is a fully realised logic-based agent that has been demonstrated in a prototype system. We explore the applicability of the KGP model for DSS, by illustrating how KGP agents may be instantiated to bring the power of agency to a DSS application. As shown in Figure 5, the basis of the KGP model is the *knowledge* of the agent. This is accessed by a modular collection of *capabilities* that enable the agent to plan or react, decide new goals, reason temporally and sense the environment in order to check whether goals or (for cautious agents) the preconditions of actions in plans are satisfied. Capabilities are synthesised in a collection of *transitions* that describe how, given inputs from the environment, the internal state of the agent changes. Changes of state may also occur by the agent observing either actively, in order to test whether something in the environment holds, or passively, as a result of being situated in a specific environment. Changes may also result because of the agent choosing to execute actions, sense the environment, introduce new goals or plans, revise these goals and plans, or simply react to changes in the environment. The transitions are integrated within dynamic flexible *cycle theories* that specify declaratively how the transitions are combined depending on the environment and the required behavior profile of the agent.

4.1. *The internal state of KGP Agents*

The internal state of a *KGP* agent is a triple *<KB, Goals, Plan>*. The *Knowledge Base* (KB) describes the knowledge of the agent of itself and its environment[1]. It consists of separate modules supporting the different reasoning capabilities, for example, KB_{plan} contains knowledge that enables the agent to plan and KB_{GD} contains knowledge that enables the agent to decide which goals to adopt next. One part of the *KB*, called KB_0, holds the (dynamic) knowledge of the agent about its external world, including what the agent has observed, the actions it has executed, and communications it has received from other agents. We assume that KB_0 is the only part of the KB that changes over time.

The *Goals* of an agent is a set of properties that the agent has decided that it wants (desires) to achieve at given times, within certain temporal intervals (specified by temporal constraints). *Goals* are split into two types: *mental goals*, that can be planned for by the agent, and *sensing goals*, that can only be sensed (to find out from the environment whether they hold or not).

The *Plan* of an agent is a set of partially ordered (atomic) actions through which the agent intends to satisfy its goals. Actions are split into three types: *physical actions*, that the agent can execute to effect a change in the external world, *communicative actions*, used by the agent to exchange information with other agents (e.g. to make a request), and *sensing actions*, for obtaining information from the environment. Actions are to be executed at given times, within temporal intervals

[1] It is also possible to represent the beliefs about other agents, but this issue is beyond the scope of this work.

specified by temporal constraints (as for goals). Note that actions are added to the *Plan* via the capabilities of the agent, which we describe in the next section.

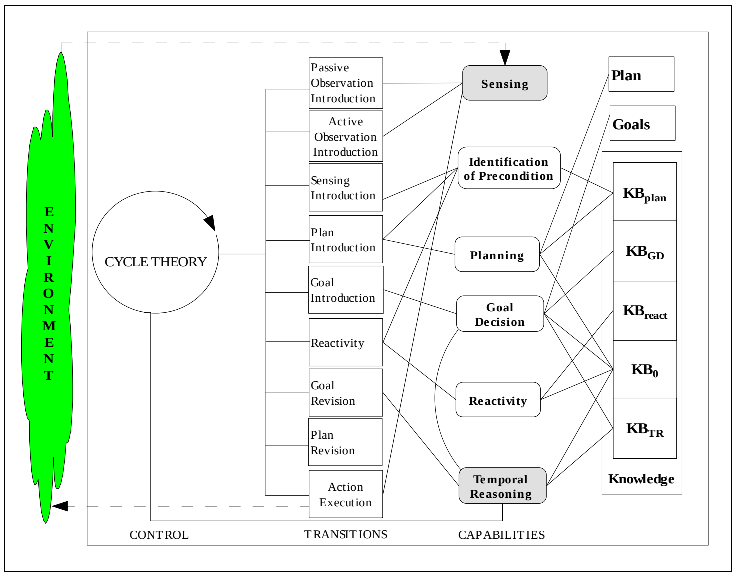

Figure 5. The KGP model of agency and its components.

4.2. *Capabilities*

The KB part of the state of a *KGP* agent is accessed by a set of *reasoning capabilities* that are specified in computational logic. These capabilities enable the agent to perform different reasoning tasks, as described below.

– A *Planning* capability uses KB_{Plan} and KB_0 to generate *partial plans*, i.e. sets of actions and (sub)goals, for a given set of goals. This allows the agent to be *adaptable* to changes in the environment without wasting planning effort.

– An *Identification of Preconditions* capability uses KB_{Plan} to identify the preconditions for the successful execution of given actions. This capability is needed for “cautious” agents, which might want to sense the environment prior to execution of actions to make sure that the execution will be successful. In addition, by means of this capability an agent may realise that the required preconditions of some actions will never be satisfied, and thus those actions may be dropped from the current *Plan*, thus allowing for the adaptability of agents.

– A *Goal Decision* capability uses KB_{GD} and KB_0 to determine the top-level goals that the agent prefers to achieve under the current circumstances.

– A *Temporal Reasoning* capability (which is also used within some other capabilities and components of the model) uses KB_{TR} and enables the agent to reason from observations stored in KB_0 and make predictions about properties on the basis of these observations. It plays a fundamental role in rendering the agent *adaptable* to changes in its environment by dealing appropriately with *partial information*, which evolves over time.

– A *Reactivity* capability allows the agent to react to changes in the external environment. Reactivity uses KB_{react} and KB_0 to identify which new actions and new goals should be added to the current state of the agent in the light of the new observations perceived in the environment.

– A *Sensing* capability allowing them to perceive the environment in which they are situated.

The list above represents the foundational set of capabilities comprising a KGP agent. These are then employed within *transitions*, which we describe next.

4.3. *Transitions*

Transitions are viewed as rules whose application changes the agent's internal state, as outlined below.

– *Passive Observation Introduction* (POI) changes KB_0 by introducing unsolicited information coming from the environment or communications received from other agents. Calls the Sensing capability.

– *Active Observation Introduction* (AOI) changes KB_0 by introducing the outcome of sensing actions for properties of interest to the agent; these properties are actively sought. Calls the Sensing capability.

– *Sensing Introduction* (SI) transition adds to the current *Plan* new sensing actions for sensing the preconditions of actions already in *Plan*, and uses the *Sensing* capability.

– *Plan Introduction* (PI) changes part of the *Goals* and *Plan* of a state, according to the output of the Planning capability. This transition uses also the Identification of Preconditions capability, in order to equip each action *A* in the set *As* computed by Planning, with the set of preconditions for the successful execution of *A*. However, this does not necessarily mean that such preconditions will be checked at the time of the execution of the actions.

– *Goal Introduction* (GI) changes the *Goals* of a state by replacing them with goals that the Goal Decision capability decides to have highest priority.

– *Reactivity* (RE) is responsible for updating the current state of the agent by adding the goals and actions returned by the Reactivity capability. As with PI, this transition too uses the Identification of Preconditions capability, in order to equip each action *A* in the set *As* computed by Reactivity, with the set of preconditions for the successful execution of *A*.

– *State Revision* (SR) revises *Goals* and *Plan*, e.g. by dropping goals that have already been achieved or that have run out of time and by dropping actions that have already been executed successfully or that have run out of time, by using the Temporal Reasoning capability and by checking the temporal constraints of the goals and actions.

– *Action Execution* (AE) is responsible for executing all types of actions, thus changing the KB_0 part of *KB* by adding evidence that actions have been executed. Calls the Sensing capability for the execution of sensing actions.

The operation of an agent is then given by the application of transitions in sequences, which produce progressive changes over the state of the agent. In KGP these sequences are not determined by a one-size-fits-all cycle of behaviour, as in BDI architectures (Rao *et al.* 1995), but rather by reasoning with a flexible *cycle theory*. Such a theory is sensitive to changes in the environment and the internal state of an agent and at the same time provides a means of declarative and intelligent control. Such declarative control allows the developer to build (behaviourally) heterogeneous agents based on different cycle theories (Kakas *et al.* 2004).

4.4. *KGP agents in a DSS context*

We illustrate the usefulness and use of the KGP model for agent-based situated decision support in the context of an e-shop, whereby KGP agents can provide guidance and help to users that need to decide on purchasing books and theatre tickets over the Internet. We will provide simple examples of the functioning of the KGP model in this setting, focusing on a high-level description and abstracting away many of the specific technicalities. The examples will be formulated by using natural language, but this formulation maps naturally to a computational logic representation (Stathis *et al.* 2005). We will make use of the KGP components shown in Table 4 below.

Transitions	**Capabilities**
Goal Introduction (GI)	Goal Decision
Plan Introduction (PI)	Planning
Reactivity (RE)	Reactivity
Passive Observation Introduction (POI)	Sensing
Action Execution (AE)	

Table 4. *KGP transitions and capabilities used in the example DSS context provided in this section.*

For simplicity, we will also abstract away from the cycle theory in the examples that follow and focus on the knowledge representation and reasoning employed by KGP.

4.4.1. *Example 1: KGP Agents for e-shops*

In this simple example we assume that a single KGP agent is assigned to support a specific user or, more generally, a specific class of users in the context of e-shops for books and theatre tickets. To support a specific user in such an application, an agent must be in a position to generate goals relevant to the domain according to the needs of the user. For this purpose we assume that the agent is developed to contain user-specific domain knowledge for KB_{GD}, for example:

Rule 1: the user prefers purchasing theatre tickets to purchasing books if there is a theatre festival.

Rule 2: the user prefers purchasing books to purchasing theatre tickets.

Rule 3: the user likes to take advantage of any special offer, both for books and theatre tickets.

Rule 4: Buying books is incompatible with buying theatre tickets.

This knowledge is used to reason about the current needs of the user, depending on their current situation and the current state of the environment in which the user is located.

To illustrate how this knowledge is utilised by the model, assume that at some time the agent observes the environment by executing a POI transition at that time, and that there is a special offer on some book *b* at some site *s*. Then, after executing a GI transition, it would introduce the goal to inquire about the purchase of *b* from *s* (referred to as goal *g* below). Indeed, *Rule 3* in KB_{GD} would force such a goal on the agent. The POI transition might consist in reading an e-mail or in noticing a change on the web page of the site having the special offer, via the agent's sensing capability.

For the purposes of the example we will also assume that the agent has the following knowledge in the Planning module KB_{plan} of its KB:

Rule 5: In order to inquire about the purchase of a book b from site s, one needs to find out the price p of b at s (referred to as action a_1) and find out the availability of b at s (referred to as action a_2).

Moreover, we will suppose that the reactivity module of the agent's KB_{react} contains the reactive rules:

Rule 6: If the agent has goal g and the price p of b is within the user's budget and b is available at s, then notify the user about buying b for p at s.

Rule 7: If the user instructs the agent to buy a book from a site, then the agent should go ahead and purchase such book.

Given this information, when the agent executes a PI transition, the goal *g* will be equipped with a plan consisting of actions a_1 and a_2. Then, AE will execute these actions (in sequence or in parallel). Suppose that the execution will instantiate *p* to 10 and confirm availability of *b* at site *s*. Then, by RE with reactive *Rule 6* above, assuming that the user's budget is 100, the agent will generate the further action a_3 to notify the user, and AE will perform the actual notification by executing a_3.

Assuming at this stage that the agent observes, via POI, that the user is happy with the purchase and is instructing the agent to perform it, then, via RE with reactive *Rule 7* above, the agent will introduce action a_4 to purchase *b* at *p* from *s*, and via AE the agent will perform the purchase on behalf of the user. Note that the user might decide instead to not purchase the book or to purchase it autonomously. It can notify the agent of its decision and the agent can take this into account by storing it in KB_0 and by reasoning on this information. Thus, KGP agents can *aid* decision-making, but not be responsible for taking all decisions on behalf of users.

Note that in more complex examples, plans might be hierarchically structured and some of the decisions might actually correspond to complex plans rather than directly to actions as in the example above. Moreover, in a more complex context, actions a_1 and a_2 above might actually stand for meta-actions that need sequences of lower-level actions to be executed. For example, they might require the provision of a login name, a password etc. Yet in other settings, they might require partaking in an auction.

4.4.2. *Adding communication*

This is a variant of the earlier example showing how multiple KGP agents, aiding decisions for different users, might need to interact with one another. The setting is the same as for example 1, but here, the user’s KGP agent U_1 needs to check with the KGP agent U_2 of another user (or example, the agent of another member of the user’s family) in order to find out whether there are any plans to purchase the same book of interest. Indeed, in this example, U_1 also contains in its

KB_{plan} the provision that it needs to communicate with U_2 before checking the chosen site and thus proposing the book to the user.

4.4.3. *Example 3: Taking decisions about the user*

This example demonstrates joint decision-making, critiquing and taking decisions about the user. In the previous examples, the system took decisions about the domain on behalf of the user. This example demonstrates the KGP agent taking decisions about the user by making choices about which conditions the user is alerted to in the course of solving an entertainment problem. It also shows simple collaboration between the system and user in problem-solving. The agent KB includes the previous rules 1, 2 and 3 in addition to the following defined rules.

Rule 8: if the user suggests a goal, then adopt as a user goal.

Rule 9: the user prefers Stacy's over all other bookstores.

Rule 10: in order to fulfil a goal to buy a book, the agent needs to choose a bookshop and then actually purchase the book from it.

Rule 8 would belong to KB_{GD} and rule 9 to KB_{plan}. After a goal request from the user to buy a book for entertainment tonight, then using rule 8, the goal is adopted (via the GI transition). Using rule 9 and 10, the plan to realise the goal buy a book is created (via a PI transition): go to Stacy's, purchase book.

Rule 11: visiting Stacy's is incompatible with visiting the theatre.

Rule 12: there is a good theatre play

Rule 13: if a good theatre play is to be missed then inform user

If KB_{plan} also contains rules 11 and 12, while KB_{react} contains 13, then, if the earlier plan is followed, then there is a consequence that a good theatre play will be missed. It is considered important to inform the user about this, because the user's preferences have been violated. Therefore using rules 1, 2, 3, 11, 12 and 13, by RE, the agent decides to inform the user of the consequence: you will miss a good theatre play.

Rule 14: if the user rejects an alternative, then search for more solutions.

The user may then use their world expertise to advise the system, to look for an alternative to the action “go to Stacy’s bookstore”. By rule 14’s initiative, interrogating external databases, the system found a solution meeting both goals: go to Stan’s, buy book, go to theatre.

5. Decision Making Processes based on KGP Agents

In this section we recap in more detail how the use of the KGP model supports agent decision making, as well as how multiple agents can support decision-making processes. As discussed in section 2, (Fischhoff 1986) and (Yates 2003), “deciding” involves many complex cognitive notions. We believe a model of agency should be specifically engineered to support this challenging type of application. In this section we interpret some of these notions in KGP terms and discuss the implications the interpretation has on the design, specification, and implementation of decision-making multi-agent systems.

5.1. *Decisions in KGP*

In the KGP model, a decision *process* is modelled at the level of cycle theory. A decision process in KGP is then an application of a sequence of transitions that will result in the generation of either a new set of goals (using GI), a new set of actions (a plan) for an existing goal (using PI), or a new set of actions by reaction to an incoming observation (using RE). These transitions will typically be followed in the cycle theory by the execution of one or more of the actions (using AE). Prior to the execution of the first action, there is a sequence of transitions defined in the cycle theory that update internal structures and record information relevant to the decision. The cycle theory therefore results in a complex sequence of events closely mirroring the three step decision process ending in the “moment of choice” as described in (Hoffman *et al.* 2005). Repeated application of transitions will in turn deal with further decomposed goals or lower-level decisions, if required.

There are a number of observations that can be made for the above cognitive interpretation of decisions. Firstly, *deciding* in the KGP model involves instantiating either goals using the goal decision knowledge base (KB_{GD}) or plans for specific existing goals. Secondly, chosen actions are intended to bring about states of affairs that serve the interests and preferences of particular agents, including one’s self (Yates *et al.* 2006). Thirdly, commitment to act according to a given goal subject to planning activity in KGP is distinguished from the execution of the action. This distinction is apparent when not all candidate actions are executed, as happens when an action of a plan remains in the knowledge base of the agent, but has timed-out. While the cognitive view of decision representation in KGP still accurately reflects the internal representation, it has the additional advantage that decisions can be justified in high-level, user terms, when necessary.

5.2. *Cardinal Issues for the Decision Making Process*

A decision-making process involving multiple agents and users can be very rich, therefore we focus our discussion on the cardinal factors identified and elaborated in section 2, (Yates 2003), and (Yates *et al.* 2006). We particularly argue that these factors complement the KGP's underlying decision making architecture by suggesting ways in which the various KBs of KGP agents should be programmed.

The first factor is the need to decide. This addresses the question of why we are (or not) deciding anything at all. Needs can be represented internally in the knowledge of an agent, in particular as conditions in KB_{GD} or KB_{RE} rules. These in turn generate new goals and plans, and as a result lead to the execution of actions. Needs can also be extracted from observing the environment and may act as triggers to rules about goals and actions. This may further identify threats and opportunities, first for the agent itself, or for other agents, and in particular, the user whose interests the agent is aiming to support. Some needs can be represented with straightforward logical rules, however for others, the agent might need to analyse a situation using more traditional decision theoretic techniques to identify whether there is a significant decision problem to solve in the first place.

The second factor is the mode of the decision, viz. who must decide and how. To identify circumstances where there is need for a decision is insufficient. KGP agents must be programmed with a model of authority structure for supporting the decisions for a specific task within a multi-agent DSS. This includes seeking the opinion of specifically selected other agents. In (Yates 2003) the authors argue that seeking opinions from others is often useful between humans, but little is known about how people evaluate and aggregate the feedback they receive. Using communication of opinion, we can also address *value*, that is, how much other agents care for a decision we need to take.

The third factor is generation of plausible options. In KGP, options are created as a plan of actions, generated by plan introduction or reactivity. The actions are selected to be a plausible subset of the actions the agent is capable of performing. These are selected using an interpretation of KB_{plan} and KB_{RE} at a specific point in time. Furthermore, the action selection function mechanism recognises that only a subset of these plausible actions are best choices. This system implicitly acknowledges that considering all possible options can be computationally wasteful. KGP does not support the evaluation of all possible alternatives for a specific interaction stage; this has to be programmed specially in the conditions of the rules.

All actions have payoffs and costs. KGP makes use of preference reasoning, as present in KB_{GD} theories, to allow agents to select actions compatible with their preferences. However, trade-offs between preferences are complex, as are the relations to external factors such as the acceptability of a decision to other agents, and how the decision is to be implemented. The acceptability of decisions together with factors of “judgement” require that the model be extended to cater for these extended factors in a wider context.

5.3. *Implementation of the Decision Making Process*

Once we have taken a decision, it needs to be implemented. We have seen in previous sections, in particular 4.1, that implementation of a decision is achieved using the AE transition. This transition will implement a decision only if the constraints of the action that implements the decision (e.g. time constraints) are satisfied. For certain types of agents, such as *careful* agents, the preconditions of an action will also be checked before the action is carried out. Commitment to act does not necessarily have an action as its primary functionality. In some circumstances, a decision might involve the successful implementation of a whole plan, with sub-plans, representing sub-decisions. KGP allows complex decisions of this kind to be implemented using the PI and AE. It can also interleave plan execution and observation and can backtrack by exploring alternative sub-plans to achieve specific decisions expressed in terms of goals.

We distinguish between implementing a decision that is a plan of an individual agent from a plan that is agreed between many agents. KGP may be programmed to have agents organised accordingly so that one agent decides the plan and the others execute it. However, to fully realise this, we need to define communication protocols for collective plan executions, including how individual agents will be responsible for the achievement of the final result. This is the subject of future work.

6. Related Work

The Retsina agent architecture (Sycara *et al.* 1996) defines three type of agent classes: interface agents, task agents and information agents. The principle role of task agents is to formulate plans for problem solving. Information agents provide access to heterogeneous collection of information sources. Interface agents provide a bidirectional communication with the user, including collecting relevant information to initiate a task, presenting results and explanations, asking the user for additional information during problem solving, and asking for user confirmation or approval, when considered appropriate. Retsina has been applied to aircraft maintenance decision support (Shehory *et al.* 1999).

(Cuppari *et al.* 1999) uses multi-agency as a logical simulation tool to model train traffic. The ability to query the logical model provides the information to support decisions of a user. However, the system is not an agent-based DSS in the sense that it participates in the user decision process. It is a conventional DSS using a multi-agent logical simulation model of an environment to supply information to a user’s decision process.

An alternative way for the system to participate in the decision process has been realised by competing arguments in CAPSULE (Krause *et al.* 1995), (Fox *et al.* 1998) designed by the Imperial Cancer Research Fund. Each argument presents a case for, or a case against a possible procedure. This is a very robust way of conducting the required asymmetric communication with a user and thus interacts at an appropriate level.

Although related to agent design for an application other than DSS, (Lee *et al.* 2004) makes use of user satisfaction feedback given in response to an agent's decisions to elicit a user's preferences about wireless network access choices. The agent also gives feedback to the user about critical (but hidden) parameters of predicted interest. In related work, (Faratin *et al.* 2003) uses a Markov Decision Process to dynamically model the user. The method is distinct from more standard off-line preference elicitation methods in that it functions on-line incrementally improving the model of the user.

(Ossowski *et al*. 2004) creates a DSS containing a multi-agent model designed for a traffic management domain. There are many methods suggested and incorporated for handling the scalability required for larger domains. This architecture identifies in finer detail the differing social roles of structured decision making. It separates agents into six classes, comprising data agents, action implementation agents, management agents, coordination facilitators, user interface agents, and peripheral agents.

7. Discussion and Conclusion

This paper has analysed how autonomous problem solvers in the form of logic-based agents, and KGP agents in particular, can contribute to a DSS that exhibits enhanced capability over that normally found in a DSS. This enhancement is due to agent design contributing situated and structured problem-solving functionality to the DSS, such that it is able to dynamically synthesise tasks for information extraction, problem analysis and solution creation. It may also proactively frame decisions and conduct certain operations for the benefit of the user, without the need for the user to be involved in every sub-step.

The KGP agent's representation of preference information has been used as the basis for autonomous decision-making about the domain, the decision process and about the type of user interaction suitable for the circumstances. This enables a KGP-DSS to aid a greater proportion of the user's total problem-solving task.

This DSS application of agents has been revealed to be more sensitive to the communication and collaboration capability of each agent compared to a more usual multi-agent application. Part of the reason for this is that communication with a high-level reasoner (a user) is more demanding than communicating with a peer agent because it introduces substantial asymmetry and potential ambiguity into the dialog. Some of the techniques employed to solve this problem for user and system communication in the DSS application may be useful in multi-agent system design in general, providing a richer communication dialog capability between agents.

The decision-making process model possible using KGP-DSS is more general than the classical four step decision phase model of information acquisition, interpretation and alternatives extraction, and selection of the chosen case followed by decision implementation. However, it remains the subject of future research to fully realise the extent of the opportunities available to aid decision-making by the inclusion of rational agents.

There is also potential to generalise user collaboration by employing argument between the system and user. It would then be possible for a user to attack assumptions upon which a conclusion is based, or rebut a conclusion. Moreover, the system may use an argument as a form of deliberate user education about the circumstances of a decision.

8. Bibliography

Angehrm A. (1993), "Computers that Criticize You: Stimulus-Based Decision Support Systems," *Interfaces* 23: 3-16.

Bonczek R.H., Holsapple C.W., Whinston A.B. (1980), "Future Directions for Developing Decision Support Systems," *Decision Sciences* 11(1): 616-631.

Cuppari A., Guida P.L., Martelli M., Mascardi V., Zini F. (1999), *An Agent-Based Prototype for Freight Trains Traffic Management*, Proc.~of FMERail Workshop 5, Toulouse, France, Springer-Verlag.

Faratin P., Lee G., Wroclawski J., Parsons S. (2003), "Social User Agents for Dynamic Access to Wireless Networks," *Proceedings of AAAI Spring Symposium on Human Interaction with Autonomous Systems in Complex Environments*: 52-59.

FIPA (2002), "Interaction Protocol Specifications," *http://www.fipa.org/repository/ips.php3*.

Fischhoff B. (1986), *Decision Making in Complex Systems*, Intelligent Decision Support in Process Environments, Springer-Verlag.

Fox J., Thomson R. (1998), "Decision support and disease management: a logic engineering approach," *IEEE Transactions on Information Technology in Biomedicine* 2(4): 217-228.

Hoffman R., Yates J.F. (2005), "Decision(?) Making (?)," *IEEE Intelligent Systems*: 76-83.

Ignizio J.P. (1991), *An Introduction To Expert Systems*, McGraw-Hill.

Kakas A.C., Mancarella P., Sadri F., Stathis K., Toni F. (2004), *The KGP model of Agency*, Proceedings of the 16th European Conference of Artificial Intelligence, Valencia.

Keen P.G.W., Scott-Morton M. (1978), *Decision Support Systems: An Organisational View*, Reading, Mass, Addison-Wesley Publishing.

Krause P., Ambler S., Elvang-Goransson M., Fox J. (1995), "A Logic of Argumentation for Reasoning under Uncertainty," *Computational Intelligence* 11: 113-131.

Laurel B. (1990), *Interface Agents: Metaphors with Character*, The Art of Human Computer Interface Design, Addison-Wesley.

Lee G., Faratin P., Bauer S., Wroclawski J. (2004), "A User-Guided Cognitive Agent for Network Service Selection in Pervasive Computing Environments," *Second IEEE International Conference on Pervasive Computing and Communications*: 219.

Mankins M.C., Steele R. (2006), "Stop Making Plans; Start Making Decisions," *Harvard Business Review*.

Matsatsinis N., Moraitis P., Psomatakis V., Spanoudakis N. (1999), *An Intelligent Software Agent Framework for Decision Support Systems Development*, European Symposium on Intelligent Techniques (ESIT), Chania, Greece.

Mora M., Forgionne G., Gupta J., Cervantes F., Gelman O. (2003), *A Framework to Assess Intelligent Decision-Making Support Systems*, Knowledge-Based Intelligent Information and Engineering Systems, Springer.

Morton M.S. (1971), "Management Support Systems, Computer Based Support for Decision Making," *Division of Research, Harvard University, Cambridge Massachusetts*.

Ossowski S., Cuena J., Kakas A., Mancarella P., Sadri F., Stathis K., Toni F. (2004), *Declarative Agent Control*, Proceedings of the 5th International Workshop on Computational Logic in Multi-agent Systems (CLIMA V), Lisbon.

Pitt J.V., Mamdani E.H. (1999), *A Protocol-Based Semantics for an Agent Communication Language*, Proceedings of the Sixteenth International Joint Conference on Artificial Intelligence, Morgan Kaufmann.

Power D.J. (2002), *Decision Support Systems: Concepts and Resources for Managers*, Quorum Books, Greenwood Publishing.

Radermacher F.J. (1994), "Decision Support Systems: Scope and Potential. Decision Support Systems," *Decision Support Systems* 12(1): 257-265.

Rao A.S., Georgeff M. (1995), *BDI Agents: from theory to practice*, Proceedings of the 1st International Conference on Multiagent Systems, San Francisco, CA.

Shehory O., Sycara K., Sukthankar G., Mukherjee V. (1999), *Agent aided aircraft maintenance*, Proceedings of the Third International Conference on Autonomous Agents (Agents'99), Seattle, WA, USA, ACM Press.

Simon H. (1960), *The New Science of Management Decision*, Harper & Row, New York.

Sprague R.H., Carlson E.D. (1982), *Building Effective Decision Support Systems*, Prentice-Hall publishers.

Stathis K., Toni F. (2005), *The KGP Model of Agency for Decision Making in e-Negotiation*, Proceedings of the Joint Workshop on Decision Support Systems, Experimental Economics, and e-Participation, Graz, Austria.

Sycara K.P., Pannu A., Williamson M., Zeng D., Decker K. (1996), "Distributed Intelligent Agents," *IEEE Expert* 11(6): 36-46.

Turban E., Aronson J.E., Liang T.-P. (2006), *Decision Support Systems and Intelligent Systems*, Prentice Hall.

Yates J.F. (2003), *Decision Management: How to Assure Better Decisions in your Company*, Jossey Bass.

Yates J.F., Tschirhart M.D. (2006), Decision Making Expertise, *Cambridge handbook of Expertise and Expert Performance*, Ericsson K.A., Charness N., Feltovich P.J.Hoffman R.R., Cambridge University Press.

Zambonelli F., Jennings N.R., Wooldridge M. (2003), "Developing Multiagent Systems: the Gaia Methodology," *ACM Transactions on Software Engineering and Methodology* 12(3): 317-370.