GoalSPEC: a Goal Specification Language supporting Adaptivity and Evolution

L. Sabatucci[1], P. Ribino[1], C. Lodato[1], S. Lopes[1], and M. Cossentino[1]

ICAR-CNR, Consiglio Nazionale delle Ricerche, Palermo, Italy
{sabatucci,ribino}@pa.icar.cnr.it
{c.lodato,s.lopes,cossentino}@pa.icar.cnr.it

Abstract. The characteristic of being autonomous and proactive makes the agents able to explore a wide solution space, that dynamically changes or contains uncertainty. We propose a language for describing system goals that may be injected at run-time into the system. The novelty of our approach consists in decoupling the business goals (what is expected) and their implementation (how to address the desired behavior). Indeed relieving the tension between 'what' and 'how' provides more degrees of freedom to the system. On the occurrence, agents of our system may exploit their features (mainly autonomy and proactivity, but also learning and planning) for getting benefits from a wider solution space. The result is that the system behavior may adapt to the current operating conditions. Moreover, the injection mechanism contributes to reduce the effort in evolving the system. This paper focuses on the goal specification language that is the base for enabling both adaptivity and evolution.

1 Introduction

The current work arises in the context of the project Innovative Document Sharing (IDS) [1], whose aim is the development of an adaptive and autonomous workflow enactment engine for improving task coordination and document management in small and medium local companies. The project exploits the well-known BPMN standard [1], among its assets, because the system will be used in real business contexts. Indeed the BPMN is mainly targeted to humans, being very flexible and expressive and it includes the notation for describing workflows as orchestration of both automatic services and human tasks. Moreover, the business domain is a highly variable application context. Business rules could change very frequently due to the evolution of business strategies, to the change of company short/middle term goals, or due to the dynamic society with its laws and regulations that must be respected. The BPMN does not support a dynamic context. Every external change must be implemented into the workflow as a set of modifications. In other terms, the workflow must be re-designed for implementing any new requirement, checking inter-dependencies and verifying the validity of the result.

It is a matter of fact that the task of designing and evolving business model is not trivial: a great number of malfunctions in workflow systems depend on business analysis errors [2–4]. Adopting a workflow system able to autonomously react to changes of the

[1] The IDS project is funded by the Autonomous Region of Sicily (POR FESR Sicilia 2007-2013)

context may simplify the work [2]. An adaptive workflow is conceived as a normal workflow but it is also able to react to some changes in the environment [2]. The need for self-adaptation is often linked to the need of reacting to exceptions [5]. Whereas BPMN already provides mechanisms to specify how the system will react to expected exceptions [1], it is more interesting to define how to react to unexpected exceptions. Indeed these events can not be handled by traditional workflow engine. For all these reasons, self-adaptation is particularly desirable for a workflow system.

In last years, self-adaptation has been gaining more and more attention specially in agent-oriented software systems [6–8]. It is a fact that multi-agent systems encapsulate an adequate level of abstraction useful to implement software capable to react to changes. Agent autonomy makes the system able to modify its behavior without supervision. The agent ability to perceive its environment helps to monitor parameters of the context that may variate. Finally agent pro-activity and the reasoning ability help to plan the appropriate reaction strategy according to agent's goals.

Generally agent goals are an higher level of abstraction with respect to the programming language (for instance in Belief-Desire-Intention systems), so that they disappears into agents' code. More recently, an interesting feature of agent is the self-awareness, that is the ability of agents to know its capacities and its goals. It is the direction of works like that of Morandini et al. [7], or that of Buhler and Vidal [6], in which the agent selects the most appropriate behavior by reasoning on a goal-model. Goal-models [9–11] have been a great advance in requirement engineering, because they provide the adequate level of abstraction to reason on the domain, its inhabitants and their needs, translating all this in a precise set of requirements. It has been proved that self-adaptive systems may benefits from relaxing the rigid constraints that is typical of traditional requirement engineering [12–14]. Intuitively a system that must overcome an obstacle must have space to change direction. A rigid set of system requirements could not provide enough space to move around a possible obstacle.

The proposed approach consists in relaxing the link between what is expected the system do (system goals) and how the system is expected to do that (system capabilities and plans). By decoupling these two aspects it is up to the system to know how to match a specific capability with the desired result. This responsibility may be satisfied only if 1) the software is aware of its goals and its capabilities, 2) it is able to reason on how to compose its behavior and 3) it is autonomous to operate without any supervision. Multi-agent systems naturally offer all these characteristics. Such architecture requires a specification language that enables this decoupling. To the best of our knowledge, none of the existing languages for goal specification is completely suitable for our scope. This paper focuses on GoalSPEC, the proposed language to express system requirements in a form that supports adaptivity and system evolution.

The remaining of the paper is organized as follows. Section 2 describes the motivation of our work, and Section 3 presents a literature review of a selection of existing goal-oriented languages. Section 4 describes the characteristic of GoalSPEC and its application to the IDS project domain. Some discussions and final conclusions are presented in section 5.

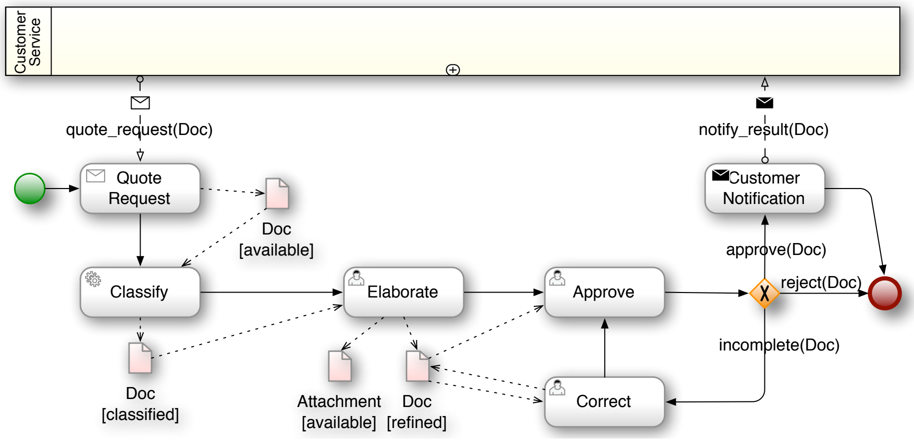

Fig. 1. Example of BPMN diagram from the IDS case study.

2 Motivation

The IDS project aims at developing a workflow enactor system for improving task coordination and document management in small and medium local companies. The project is a benchmark for exploring real motivations since a workflow system shall autonomously configure its behavior. In particular the requirements elicitation phase of the project highlighted that the business domain is a high dynamic context, in which company business goals and rules often vary. In addition the workflow engine runs in a socio-technical context in which human and social factors are relevant. In particular such systems operate in a society whose laws and regulations are frequently revised and modified.

The business process in Fig. 1 models the process for quote request in a generic company. The customer service receives requests from customers via telephone, fax, email or traditional mail. The flow starts when the customer service fills a quote request form. This generates a virtual document to be processed to satisfy the request. The automatic task *Classify* tries to identify the input request type, when possible, by using image processing techniques. Therefore, the classified document is manually elaborated (*Elaborate* task) by the technical responsible who produces the quote as virtual attachment. It is the commercial manager's turn to supervise the attachment and to mark it as approved, incomplete or rejected (*Approve* task). If the document is marked as incomplete, a new revision loop is activated (*Correct* task). Conversely, if the document is approved then the customer notification is responsible to contact the customer and to provide the requested information.

The objective of this paper is to discuss how decoupling business goals and their implementation for self-adaptation purposes. Whereas BPMN is extremely flexible to define the implementation layer, it lacks of an explicit syntax for defining business goals. We look at a goal specification language with the following characteristics.

REQ.1 - *The language shall be powerful enough to represent requirements and constraints for information systems*. System requirements describe the expected results of the system and they can be articulated into functional and non-functional requirements. Whereas functional requirements describe the behavior in terms of the expected functionalities, non functional requirements generally describe the expected performances of the system. This feature is central for our purposes since we want to model BPMN goals as expected behavior of the workflow system. An important aspect to consider is the massive presence of social factors into the system behavior: system constraints are norms that specify what the system is obliged to do and what is forbidden.

REQ.2 - *The goal specification shall be independent on how the software system will work*. It is out of the scope of the language to describe how to address the specified goals. The focus must be on 'what' is expected, so the language shall be in a declarative fashion. We work under the hypothesis that the portion of the world under interest is decomposed and described by a set of states and properties. For our purposes, describing system goals means to describe the expected result by grounding it on ontological bases.

REQ.3 - *The language shall be context-free*. Agents are the main consumers of the goal specifications, because they are responsible to adapt their behavior to the goals. Goals are not statically defined at design-time automatically, but can be modeled or modified after agents' life begins. Therefore, for our purposes, the goal language must be interpreted at-run time by system agents, who must acquire the expected result that is encapsulated in them. Agents that are aware of the expected state of the world (that is desired by humans) can reason on how to adapt their behavior according to perceptions and to desired goals.

REQ.4 - *The goal specification shall be compatible with the expressiveness of the BPMN language*. The Business Process Model and Notation is a very expressive graphical language that is able to express almost every process. It includes, among the others, human and software collaborations, conversations and choreography, concurrent task execution, task decomposition, persistent data, handling of error/-compensation/escalation situations. We work under the hypothesis that in a process every task is done because underlying goal exists, and we want to automatically extract these goals every time this is possible. The resulting goal must perfectly synthesize its task, so the specification language must be expressive enough to cover the whole specifications of BPMN.

REQ.5 - *The language shall be attractive for a business audience*. Goals will be automatically extracted (when possible) from the BPMN description of the business process. However analysts want to maintain the control of the workflow execution. For this reason they will want to verify and manually refine system goals before these are injected into the system. For our purpose, the language must be simple to learn, to understand and to use by non-technical people. We consider to use a specification language that is closer to the natural language a better choice than a formal language based on mathematics basis.

REQ.6 - *The language shall be flexible enough to include points of uncertainty in specifications*. Traditional languages for goals specification generally adopt a strict definition of the functional and non-functional requirements in order to avoid ambiguities or uncertainty. Despite this is perfect for traditional system development, this may represent a limitation for a self-adapting system. Indeed the adaptation mechanism need wider solution space where to move, in which many alternative solutions are possible, each with different trade-offs. For our purpose, we want to increase the degrees of freedom of agents in finding the solution. We want to allow the presence of points of uncertainty in the goal specification to let agents relax some constraints on the necessity.

3 Review of Goal Specification Languages

We conducted a systematic literature review, according the principles of Wohlin et al. [15]. The research question is about the expressiveness of *goals* in literature. In particular we check if they match with the characteristics defined in Section 2.

We identified 30 among the most relevant papers in the area, that are distributed according their topic in informal/semi-formal, formal approaches and implementation. The table shown in Fig.1 summarizes the results of this comparison.

Totally informal approaches commonly express goal semantics by using natural language expressions: they are similar (in our classification) to semi-formal representations that mix graphical and text based notation. These are the most frequently used techniques for specifying goals because they facilitate the exchange of knowledge with stakeholders.

Business Motivation Model (BMM) [16] is a meta-model and a standard for capturing semantically rich business requirements, useful for analysis, querying, impact analysis, change management and business reasoning. It tries to highlight "why the business wants to do something, what it is aiming to achieve, how it plans to get there and how it assesses the result". Nevertheless, BMM does not come with a standard graphical notation, it has a broader scope than just goal modeling and therefore it has too many concepts (some of which are unclear or overlap with each other), it has no strong formal basis and does not address at all goal analysis and reasoning issues. Finally its goals do not ground on ontological bases and not support reasoning with uncertainty.

The *Goal-Scenario Coupling* [17] is a language that expresses a goal by a structured natural language in which any clause has a main verb and several parameters. For example Display (the error message)$_{Obj}$(to the customer)$_{Dest}$, where each parameter plays a different role with respect to the verb. We note, among parameters, there are *means* and *manner* that define how to address the goal satisfaction.

The $i*$ [18] framework and *Tropos* [9] are semi-structured language in which goal statement are free-text but relationships are formalized. In particular And/Or Decomposition relations, means/end relationship (setting means to reach a goal), and contribution relationships (expressing positive or negative contribution to goal achievement). The language is perfectly suitable for requirement analysis, but the poor semantics of the natural language goal definition makes it hard to move towards implementation phases.

To overcame this point the *Formal Tropos* [19] extension offers all the primitive concepts of $i*$ [18] but more expressive power, and in particular, temporal specifications. As well as *Tropos* does, *Formal Tropos* describes all the relevant objects of the modeled domains along with their relationships, but it also allows to represent dynamic aspects of the model by a first order linear-time temporal logic with future and past time operators. The language offers existential and universal quantifiers for defining Constraints (which restrict the valid executions of the system), Assertion (which are expected to hold in all valid executions of the system) and Possibility (which are expected to hold in at least one valid execution of the system).

Tropos has been also used in the field of self-adaptation [7]. Morandini et al. enriched the goal model by specifying achievement conditions in relationship with the environment, and the possibility to model faults and corresponding recovery activities. They set system goals as invariants, whereas variation points of the global behavior are granted by decomposing the main goals into trees of alternative sub-goals. The system uses advanced decision techniques to select among many alternative strategies to address the main goals. Moreover, the defined goals can be directly mapped to Jadex goals. The technique of designing the expected exceptions was already comprised into the BPMN specification but the main limitation, for our purposes, is that goals and plans are paired at design time.

GRL [20] is also based on *i**. It is a language for supporting goal and agent-oriented modelling and reasoning about requirements, with an emphasis on dealing with non-functional requirements (NFRs). In *GRL*, a goal can be either a business goal or a system goal. A business goal express goals regarding the state of the business affairs the individual or organization wishes to achieve. System goals describe the functional requirements of the information system. *GRL* is a language more suitable for the first phases of analysis. The goal specifications it provides are not suitable for agents. Moreover it not support adaptation and uncertainty factors.

Differently *KAOS* [21] is conceived to produce automated specifications of domain knowledge. The framework grounds on a formal language where goals are defined by means of real-time linear temporal logic (first-order logic with modalities referring to time), that semantically captures maximal sets of desired behaviors. Goals are classified according to some patterns (*Achieve*, *Avoid*, *Maintain*, etc...); these verbs in KAOS specify a temporal logic pattern for the goal name appearing as parameter. They implicitly specify that a corresponding target condition should hold some time in the future, always in the future unless some other condition holds, or never in the future. This expressivity makes the language suitable for formal proof of specification correctness.

Winikoff *et al.* [8] present a way to integrate declarative and procedural views of goals in agent systems. They propose a plan notation called CAN (Conceptual Agent Notation) along with a formal semantics expressing both goal aspects. In this paper a goal is represented by means of two logical formulae about the agent's beliefs representing the declarative aspects and a set of plans representing the procedural aspects of goals. This kind of goal representation allows to capture several goal proprieties (such as persistence, consistence, achievement etc . . .) but it does not encapsulate uncertainty factors and it is not thought for adaptation and for BPMN mapping. Moreover, it encapsulates procedural aspects that GoalSpec deliberately avoids in order to allow adaptation in our workflow enactor. Subsequently, the GOAL language [22] incorporates declarative aspects of goals in an agent system in order to allow the agent to decide what to do. A declarative goal specifies a state of the world that an agent wants to reach. Thus they do not specify how to achieve such states. A feature of GOAL is that the set of goals is not required to be consistent because not all goals have to be reached simultaneously. Goals can be achieved at some moment in the near or distant future. These features are very close to our needs, anyway whereas the GOAL language grounds on agent mental states, we need a language to define business goals that are, in fact, humans' goals. GoalSPEC should be designed to be attractive for business audience as specified in Section 2. Moreover, our language want to be also a trade off between high-level goal languages and implementation ones

Finally, there is a lot of work about agent-oriented programming languages (such as JACK [23], AgentSpeak(L) [24], Jason [25], 3APL [26] [27], Jadex [28], etc. . .) where the goals play a central role. Many of them are associated to sophisticated reasoning engines allowing to develop complex intelligent agents. But in these frameworks goals are strictly linked to plans to reach them. In our system we do not specify a particular representation of the plan. A plan could be as usual a simple sequence of agent actions, a combination of web services, a set of procedures and so on. A strength of our work lies on decoupling the declarative aspect of a goal from its procedural one allowing thus a flexible plan composition in order to satisfy a declared goal.

Table 1 summarizes the results of the review. This table provides a kind of matching between the analyzed goal languages and the requirement we need. At the best of our knowledge, we have not found one approach that fully meets all our requirements. In particular, the attempts to use Goal-based modeling for specifying adaptation do not fully satisfy what we want to realize in our envisioned framework. Therefore, the next section proposes a new language, named *GoalSpec*, that incorporates some features of the reviewed languages but also it introduces new characteristics for our purposes.

		KAOS	TROPOS /I*	Formal TROPOS	GRL	Goal Scenario Coupling	TROPOS for adaptivity	BMM	BDI Languages	GOAL
	Formal	x		x						
	Goal Type	A - I	A - I	A - I	A	A	A - I	A	I	I
	Adaptivity			x			x			
Features	Requirements/ Constraints Representation	x	x	x	x	v	x	v	v	v
	Oontological Representation	x		x						
	Context-free Grammar	x		x						x
	Uncertainty			x						
	Human Oriented		x				x	x		

Legend
x = complete matching
v = partial matching
A = Analysis
I = Implementation

Table 1. Comparison table among goal languages and features we are interested.

4 GoalSPEC: A Language to Specify System Goals

Here, we define a language for specifying system requirements and constraints. It has to be general enough to cope with several aspects that are key elements in current systems. To do this, we have incorporated some features of existing languages and we have introduced new ones in order to address specific adaptation and business issues. For the sake of clarity we refer to the whole language as an abstract package that contains two sub-languages: *GoalSPEC* that focuses on specifying expected results of the system in terms of functions, and *NormSPEC* that is a norm-based language for specifying non-functional requirements and constraints that generate a boundary where the system is limited to move. This paper focuses on GoalSPEC only, whereas the foundations of NormSPEC are yet published in [29]. The common characteristics of both GoalSPEC and NormSPEC are: (i) their grammar is a subset of the natural language; (ii) they have context-free grammar, thus to be automatically parsed and translated into machine instructions; (iii) some elements of the specifications can be relaxed by using fuzzy modifiers. The concept of system goal is central to our language. Aligned with common definitions we distinguish between business goals and system goals.

Def.1*: Business Goals are enterprise strategic interests that motivate the execution of business processes [11]. They are discovered in phase of analysis and they are useful to model a strategic view of domain stakeholders and to elicit system requirements.*

Def.2*: System Goals are described as states of the world that the system desires to achieve [9]. System goals are generally the subset of business goals that are delegated to the workflow system in order to implement some kind of automation.*

In the following, an extract of the BNF description of the GoalSPEC language is reported:

```
goal_type : social_goal | system_goal;

social_goal : SOCIAL GOAL goalname ':' trigger_condition actors_list SHALL
  ADDRESS final_state;

system_goal
: GOAL goalname ':' trigger_condition actor SHALL ADDRESS final_state;

trigger_condition : event
     | trigger_condition AND trigger_condition
     | trigger_condition OR trigger_condition
     | NOT trigger_condition | '(' trigger_condition ')' ;

event : ON date | AFTER delay SINCE trigger_condition
     | WHEN state ;

final_state : state
     | final_state AND final_state
     | final_state OR final_state
     | NOT state | '(' final_state ')';

state : predicate
     | message_sent_state
     | message_received_state ;

message_sent_state : MESSAGE predicate SENT TO actor;

message_received_state : MESSAGE predicate RECEIVED FROM actor;

actors_list : actor AND actors_list | actor;
```

```
actor : THE_SYSTEM | THE characters ROLE;
```

Social goals and agent goals. The productions of the language allow to specify system goals. The first production of the BNF describes a goal as composed by an initial triggering *condition*, a list of *actors* that are involved and a desired final *state* of the world. We consider two categories of system goals:

- *agent goals* are atomic goal, related to a specific outcome in the workflow instance; they derive from Tasks in a workflow, and they can not be further decomposed into sub goals. Addressing an agent goal produces an advancement for the achievement of the workflow.
- *social goals* are goals that are decomposable into many sub-goals. These goals derive from Processes or Subprocess in a workflow. A social goal is not necessary satisfied when all its sub-goals are satisfied. It has its own final condition to verify in order to be considered addressed.

For instance, the agent goal, for which the *Elaborate* task in Fig. 1 must be executed, is the following:

```
GOAL doc_management.g2 :
(WHEN classified(Doc) AND WHEN done(classify))
THE worker ROLE SHALL ADDRESS
((refined(Doc) AND available(Attachment)) AND done(elaborate))
```

The *actors* section is strictly related to the concept of BPMN participant. It specifies 'who' is the main responsible to address the given goal. GoalSPEC includes two different categories of participant: human roles and the system. When the actor of a goal is *the system* then the goal may be automatically addressed. On the other hand, when a human role is responsible of a goal, the system can only monitor when the goal is successfully addressed. A social goal, generally contains a list of actors that will collaborate to the workflow enactment.

Triggering Conditions. Each goal specification starts with a set of *conditions* that must hold in order to activate the goal. The BNF specifies that a condition may be a single event or a composition of multiple events. Basic events may be:

- $on < date >$, triggers when a given day arrives. The $date$ is a parameter that follows the ISO 8601 specification [30] (International standard date and time notation). Examples are 'on 1995-02-04', or 'on 2013-04-01/23:59:59'.
- $after < delay > since < event >$, triggers after an amount of time since a given event has occurred. The $delay$ parameter specifies a duration of delay according to the ISO 8601 standard. For instance '2W' means 2 weeks.
- $when < state >$, triggers when a specified state of the world becomes true: an example is 'when rejected(Doc)'. In the following, in this same section, the specification of state is explained in details.

States of the World. One of the main operative hypothesis of this work is that the portion of the world under interest is described by using states and predicates. Indeed, GoalSPEC adopts an ontological description of the world, and logic predicates play a central role in the specification of elements and their properties. The BNF indicates that each goal specification includes a *final_state* that must be true in order to declare the goal is finally satisfied. A *state* may range from a single logic predicate '*classified(Doc)*', to a composition of multiple predicates by and/or operators '*((refined(Doc) AND available(Attachment)) AND done(elaborate))*'. Also a NOT operator is included to specify the state is true when the predicate is false. Two special occurrences of state are produced by the MESSAGE keyword. These states occur when workflow messages are exchanged (incoming or outgoing). Here two examples of message states:

– *MESSAGE notify_result(Doc) SENT TO THE customer_service ROLE*
– *MESSAGE quote_request(Doc) RECEIVED FROM THE customer_service ROLE*

We used Prolog as the dialect to define GoalSPEC first-order logic predicates in a declarative fashion. This decision has been done also to be compliant with some BDI (belief-desire-intention) frameworks such as the Jason architecture [31]. Prolog predicates use atoms to represent: *(i)* particular individuals or objects (symbols starting with lowercase, or numbers); *(ii)* variables (symbols starting with uppercase) that will assume a value with the mechanism of unification; *(iii)* facts (functors followed by list of arguments), used to represent properties or relationships. Let us consider the example in Fig. 1. After the *Approve* task the document may assume three possible states: *approved(Doc)*, *incomplete(Doc)* or *rejected(Doc)*. The value unified with the variable *Doc* represents the specific instance of the working document. Matching and unification are performed along the same set of goals specification, therefore, considering the following goal:

```
GOAL doc_management.g5 :
(WHEN approved(Doc) AND WHEN done(approve))
THE SYSTEM SHALL ADDRESS
MESSAGE notify_result(Doc) SENT TO THE customer_service ROLE
```

It is worth noting that the variable of the predicate '*approved*' will be unified with the variable of the predicate '*notify_result*': in other worlds the document that will be sent to the customer service is the same that has been approved and it is the same that has been previously '*refined*' (see goal *doc_management.g2*).

Human participants. Business processes describe sequences of operations. The BPMN standard (in contrast with BPEL) allows to declare a human role as responsible of activities. Manual and user tasks are operations that are executed without (or with a limited support) of the machine. In our vision all the activities in a workflow, including the manual ones, exist in order to pursue a business goal, or in technical terms, to take the world in a desired state.

```
GOAL doc_management.g3 :
(((WHEN refined(Doc) AND WHEN available(Attachment)) AND WHEN done(elaborate))
OR ((WHEN refined(Doc) AND WHEN available(Attachment)) AND WHEN done(correct)))
THE manager ROLE SHALL ADDRESS
(done(approve) AND (incomplete(Doc) OR approved(Doc) OR rejected(Doc)))
```

Whereas goals that derive from service tasks are directly delegated to one or more agents of the system, goals of manual/user tasks can not be delegated: they must be addressed by human resources. In these cases system goals differ from business goals, indeed the role of the system is to monitor that the desired state is correctly addressed, or when this is not feasible, to generate user interfaces where human operators may notify their progresses.

Fuzzy modifiers. It is very interesting the work by Whittle et al. [13] who defined RELAX, a language for requirement specifications that uses a declarative style for specifying possible sources of uncertainty. Flexibility is obtained by relaxing the rigid 'shall' form typical of requirements and by introducing uncertainty factors. *RELAX* is based on three types of operators (temporal, ordinal and modal) to address uncertainty. The semantics of RELAX expressions (AFTER, AS EARLY AS POSSIBLE, and so on...) is formalized in terms of fuzzy branching temporal logic. The authors suggest that requirements languages for self-adaptive systems should include explicit constructs for specifying and dealing with the uncertainty inherent in self-adaptive systems.

Likewise in *RELAX* [13], the expressiveness of GoalSPEC language is extendable to let the designer relax some constraints. This is possible by using 'fuzzy modifiers' to increase the flexibility of the rigid unification operator. For instance, it is possible to relax time constraints by specifying that a goal shall be addressed AS SOON AS POSSIBLE, or AS LATE AS POSSIBLE. This is particularly useful when the system deals with many parallel goals and must optimize the global behavior by selecting the one with highest priority. It is also possible to relax measures with the following modifiers: AS CLOSE AS POSSIBLE, AS MANY AS POSSIBLE, AS FEW AS POSSIBLE. For instance it is possible specifying the number of items in a list must be as close as possible to a given threshold. A value that is close to the threshold (but not equal) will not raise an exception, as a traditional workflow engine would do, thus allowing the workflow to continue normally. The implementation of these modifiers is a work in progress and is out the scope of this paper.

4.1 Translating BPMN into System Goals

Many times in this paper we mentioned BPMN [1] and BPEL [32] as the most common specification languages for workflow. In the IDS project we adopted BPMN because of its capability to describe human tasks whereas BPEL does not support. BPMN and GoalSPEC are not in competition, but rather are complementary. Each of them has a different role in the whole architecture. BPMN is the main interface for business analysts to model their business processes. GoalSPEC is intended to model the business goals that are not explicitly expressed in the BPMN process.

An important requirement gathered in the IDS project is avoiding additional burden to the business analyst. For this reason we elaborated an algorithm that automatically extracts goals from a BPMN specification of a business process. BPMN2GoalSPEC is a Java component that translates the BPMN specification (XML) into a set of goals according to the GoalSPEC grammar. This set of goals could be then manually revised if necessary, however it represents a good starting point for the analyst.

In order to explain what is the idea underlying the extraction of goals, we can observe that a BPMN process can be seen as a graph, where nodes are Tasks, Events or Gateways, and arcs are sequenceFlows. Sub-processes are special kind of Tasks that act as container for other Tasks, Events and Gateways. We already discussed that a process generates a social goal, since it describes a protocol of collaboration among many parts. On the other hand every Task generates an agent goal that encapsulate the business objective to execute it. As a consequence a Sub-process generates both an agent goal and a social goal.

In order to generate a social goal or an agent goal we have to extract both the triggering condition and the final state for the specific goal. The triggering condition is the condition needed for activating a specific goal and the final state is the state of the world the goal indicates to be addressed. Working under the condition that both the triggering condition and the final state could be expresses by considering the states of the world, we want to measure the state at input and output of Task nodes.

We preliminary observed that:

– Gateways do not alter the state of the world. This means the input state of a gateway is equal to the output state;
– Events can be distinguished in catching and throwing. Catching events block the execution waiting that the desired input state triggers. On the contrary, throwing events proactively generate the desired state as output;
– Tasks encapsulate both the waiting/generating behaviors. Indeed a task activates when a given input condition (we call it the waiting condition) is true, whereas it generates a given state as output (we call it the generated condition).

We easily measure the waiting/generated condition by observing at dataObjects, inputSets, outputSets, dataStores and Messages that are consumed as input of a task, or that are produced as output by the task. For instance, the *Classify* task shown in Fig. 1 waits for the *Doc* dataObjects assumes the state 'available' therefore the waiting condition is *WHEN available(Doc)*. Conversely the same task produces a new state for the *Doc* dataObjects ('classified'), so the generated condition is *classified(Doc)*.

Anyway waiting/generated conditions are different from triggering condition and final state because we must also consider that a Task is immersed in a context with predecessor and successor nodes that modify its state at input and output. In general we can say that the triggering condition can be elaborated as the waiting condition plus the backward condition, whereas the final state may be elaborated as the generated condition plus the forward condition.

The backward condition is measured by looking backward at the target node following the incoming sequenceFlow arcs, whereas the forward condition is measured by looking forward at the target node following the outgoing sequenceFlow arcs. In this analysis:

– *Gateways* propagate the state in both backward and forward directions;
– *Throwing Events* block the backward analysis, whereas *Catching Events* block the forward analysis;
– *Tasks* block the propagation in both directions;
– eventual *Conditional sequenceFlows* also provide additional useful information that have to be composed with backward/forward conditions.

Finally, the Algorithm 1 describes the goal generation technique. As an example of goal generation, let us take in consideration the *Approve* task in Fig. 1.

Inferring the triggering events of a goal. The waiting condition for the goal is related to the presence of the *refined(Doc)*, where the generated condition is the predicate *done(approve)* (this is generated by default for each task to highlight the correct execution of an activity). To build the backward condition the algorithm selects the two

```
Data: the BPMN workflow graph
Result: set of GoalSPEC goals
forall the x, Task and Throwing Event in the workflow do
    let be waiting(x) the waiting event;
    let be generated(x) the generated state;
    generate a new agent goal G;
    add waiting(x) to triggering_condition(G);
    add generated(x) to final_state(G);
    forall the i, incoming SequenceFlow(x) do
        let be cond(i) the sequence flow condition of i (when it exists);
        calculate backward(i,source(i));
        add cond(i) AND backward(i,source(i)) to triggering_condition(G);
    end
    forall the j, outgoing SequenceFlow(x) do
        let be cond(j) the sequence flow condition of j (when it exists);
        calculate forward(j,target(j));
        add cond(j) AND forward(j,source(j)) to final_state(G);
    end
end
forall the p, Process and SubProcess in the workflow do
    generate a new social goal S;
    forall the z, starting event and starting task of the workflow do
        forall the i, outgoing SequenceFlow(z) do
            calculate forward(i,target(i));
            add forward(i,target(i)) to triggering_condition(S);
        end
    end
    forall the t, ending event and ending task of the workflow do
        forall the j, incoming SequenceFlow(t) do
            calculate backward(j,source(j));
            add backward(j,source(j) to final_state(S);
        end
    end
end
```

Algorithm 1: Extracting Goals from the Workflow

incoming sequenceFlows and, follows them back until reaching the source nodes: 1) *Elaborate* and 2) *Correct*. Because of both are Tasks, the condition is equal to the generated condition. Therefore they are:
backward(1,Elaborate)=done(elaborate) AND refined(Doc) AND availabe(Attachment), and
backward(2,Correct)=done(correct) AND refined(Doc) AND availabe(Attachment).
The whole triggering condition of the goal is the OR combination of these two results.

Inferring the resulting state of a goal. The generated condition of the goal is *done(approve)* by default, since there is no explicitly produced output. For building the forward condition the algorithm follows forward any outgoing sequenceFlow from the task. In this case there is only one outgoing sequenceFlow that arrives to the exclusive gateway. The gateway node does not alter the state, but it propagates in input the state that is in output. Because it is an inclusive gateway the three outgoing sequenceFlows will be in OR. The first flow is conditional (*approved(Doc)*) and arrives to a task that blocks the forward analysis. The second flow is also conditional (*rejected(Doc)*) and terminates to an end event. Finally the third flow is conditional (*incomplete(Doc)*) and arrives to a task that again blocks the forward analysis. As a consequence the forward condition is *incomplete(Doc) OR approved(Doc) OR rejected(Doc)*.

```
GOAL doc_management.g3 :
(((WHEN refined(Doc) AND WHEN available(Attachment)) AND WHEN done(elaborate))
OR ((WHEN refined(Doc) AND WHEN available(Attachment)) AND WHEN done(correct)))
THE manager ROLE SHALL ADDRESS
(done(approve) AND (incomplete(Doc) OR approved(Doc) OR rejected(Doc)))
```

The complete goals set for the book management example (Fig. 1) is the following:

```
SOCIAL GOAL doc_management :
WHEN MESSAGE quote_request(Doc) RECEIVED FROM THE customer_service ROLE
THE worker ROLE AND THE manager ROLE AND THE customer_service ROLE AND THE SYSTEM
SHALL ADDRESS ((rejected(Doc) AND (done(approve) AND NOT done(costumer_notification)))
OR MESSAGE notify_result(Doc) SENT TO THE customer_service ROLE)

GOAL doc_management.g0 :
WHEN MESSAGE quote_request(Doc) RECEIVED FROM THE customer_service ROLE
THE SYSTEM SHALL ADDRESS
(available(Doc) AND done(quote_request))

GOAL doc_management.g1 :
(WHEN available(Doc) AND WHEN done(quote_request))
THE SYSTEM SHALL ADDRESS
(classified(Doc) AND done(classify))

GOAL doc_management.g2 :
(WHEN classified(Doc) AND WHEN done(classify))
THE worker ROLE SHALL ADDRESS
((refined(Doc) AND available(Attachment)) AND done(elaborate))

GOAL doc_management.g3 :
(((WHEN refined(Doc) AND WHEN available(Attachment)) AND WHEN done(elaborate))
OR ((WHEN refined(Doc) AND WHEN available(Attachment)) AND WHEN done(correct)))
THE manager ROLE SHALL ADDRESS
(done(approve) AND (incomplete(Doc) OR approved(Doc) OR rejected(Doc)))

GOAL doc_management.g4 :
((WHEN refined(Doc) AND WHEN available(Attachment)) AND (WHEN incomplete(Doc)
AND (WHEN done(approve) AND NOT WHEN done(costumer_notification))))
THE worker ROLE SHALL ADDRESS
((refined(Doc) AND available(Attachment)) AND done(correct))

GOAL doc_management.g5 :
(WHEN approved(Doc) AND WHEN done(approve))
THE SYSTEM SHALL ADDRESS
MESSAGE notify_result(Doc) SENT TO THE customer_service ROLE
```

4.2 The Proposed Architecture

We have anticipated that social and system goals are injected into the system at run-time. This subsection provides a brief description of the architecture of the workflow

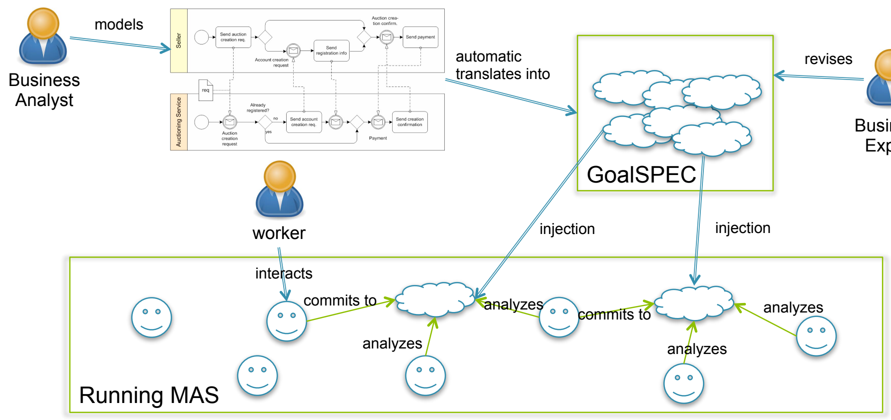

Fig. 2. Overview of the proposed system for enacting the workflow.

engine. This overview is short because of space concerns, but it helps to provide a justification to the presented language and to answer the challenge introduced in Section 2. The work assumes that, in a real working environment, BPMN is the main interface for business analysts. We decided to accept BPMN as it is prescribed by the OMG group and we avoid to create yet another extension or variation to its metamodel. Indeed, Figure 2 shows that a business analysts uses a BPMN 2.0 tool to edit the workflow. In the meanwhile, a multi-agent system is already running and its members are greedy to use their capabilities. Every agent in the system owns some specific capabilities and it is aware of them. For instance, considering the IDS project, an agent is able to classify documents, whereas a set of agents are able to communicate with the range of human roles (’customer’, ’worker’, ’manager’). When a business process is ready, it is automatically translated into a set of goals in GoalSPEC and then these goals are inserted into a database. The agents of the system detect when new goals are in the database and verify whether their capabilities are suitable to commit to one ore more goals. This triggers a social auction for assigning goals to agents. When all the agent goals are assigned to some agent, the relative social goal is activated and the workflow can be executed.

This architecture decouples the goals (the ’what’) from the capabilities (the ’how’) and it lets the agents to autonomously decide if and how to employ their capabilities to address them. The advantages are: *(i) Exploration of alternatives* - when more agents have different implementations of the same ability (for example different classification algorithms) they are in competition to get a goal assigned. Therefore, if the workflow fails, for some reason, the commitment is retreated and re-assigned, thus to explore different alternative tasks to the same objective. *(ii) Learning* - during their execution agents learn from the result of their actions (by associating the success or failure to the execution context). In this way, the social auction is won by the most capable agent according to the current execution context. *(iii) Evolution* - new goals can be injected into the database or existing goals can be retreated from it. Given that the commitment is dynamic, it is not a big deal to reorganize the agents thus to make new goals satisfied.

5 Discussion and Conclusions

GoalSPEC has been developed to cover the requirements described in Section 2, in order to implement the adaptive system shown in Fig.2. None of the existing languages meet all those requirements. In the following we make some considerations on GoalSPEC and on other works close to our approach.
GoalSPEC supports Adaptivity. GoalSPEC is intended to be used within the lifecycle of a business process from creation to maintenance. The scenario starts when business analysts generate a preliminary version of business process by employing a BPMN visual tool. The tool generates a XML file that adopts the standard schema defined by OMG. The *BPMN2GoalSPEC* component receives this file and it is able to automatically generate a set of *GoalSPEC* social and system goals. *GoalSPEC* is created in the context of adaptive workflow and it completely covers the whole BPMN expressivity (REQ. 6), hence whatever process defined with BPMN, its business goals (functional and non-functional) can be modelled with GoalSPEC (REQ. 1). Before being executed, system goals are proposed to analysts in order to be revised. Since *GoalSPEC* is based on natural language, and it is specifically been conceived to be attractive for a business audience (REQ. 4), analysts can easily understand and modify the results. This is useful since analysts may include other business goals missing in the BPMN specification.

Several time in this paper we have mentioned that social and system goals are injected into the system at run-time. Indeed, the agent system is already running when business analysts work. Agents are specialized workers (each with their specific skills) waiting for something to do. When goals are released, agents perceive them into their environment. They are also able to interpret *GoalSPEC* (REQ. 3) and to absorb goals into their knowledge base. Even if the grammar is context-free, goal specification by *GoalSPEC* is not rigid for two reasons. Firstly a goal does not specify how to operate but it rather defines the expected results in ontological terms (REQ. 2), that is the final state of the world that is desired. In addition, some elements of the behavior specifications may be relaxed by using fuzzy modifiers (REQ. 5). In practice, agents can potentially plan and propose more alternatives for addressing a goal. Social interactions and individual planning capability are out the scope of this paper.
GoalSPEC supports Evolution. Agents are allowed to commit to the achievement of injected goals as long as they are perceived into the environment. Certainly current business process will change in the future, maybe as a cause of new business goals, new laws and so on. In a traditional approach, analysts would revision their BPMN models in accordance to changes. Any revision includes to check possible inter-dependencies among related (sub)processes with a consequent hard work to ensure coherence. The *GoalSPEC* approach is that the system intelligence will support this activity. Agents ability to reason on the injected goals may also highlight possible incoherence or conflicts among them. Warning of conflicts are useful for the analysts to improve the process. The workflow system will be always running, but the consequence of a goals revision is that agents will reorganize their behavior to address the new objectives. Probably programmers will also introduce new agents into the system to cover the need for new skills.
Considerations on the Expressiveness. *GoalSPEC* adopts an ontological description of the business process. Logic predicates play a central role for the decomposition of

the domain in a set of possible states of the world. Comparing *GoalSPEC* to Tropos, it appears that the second proposes a definitively richer semantics for the relationships between goals. *GoalSPEC* does not include operators for and/or decomposition, contributions and means/end. This choice has been deliberately done in order to make agents able to automatically discover these relationships. Any Tropos relationship adds a constraint for the system working. Otherwise, system intelligence must search for alternative solutions that were not designed by analysts. Comparing GoalSPEC to KAOS, it appears that the second uses a temporal logic, definitively more expressive than first order logic. Temporal propositions, in fact, contain some references to time conditions that GoalSPEC does not support. For example, we can't specify that in the time between the event $E1$ and the event $E2$ the action A can be executed at most twice. We accept this limitation because our language needs to be more human oriented and feasible for complex systems. Indeed, systems based on temporal logic are difficultly scalable up and require formal verification. But, in order to further increase the goal expressiveness *GoalSpec* also supports some fuzzy modifiers that may introduce uncertainty with the aim to increase the agent degree of freedom in pursuing their goals.

Considerations on the Generality. The proposed language, although developed for a specific project, can also be used in more general information systems. We assert this because, it owns features that make it reusable in the general context in which workflows have to be managed. As it is well known, any information system embeds some kind of workflow even if sometimes that is not explicitly specified.

Considerations on other related approaches. Some proposals on the idea to link business processes to goal models exist in literature. To the best of our knowledge, among them those closest to our approach are presented in [33] and in [34].

G.Koliadis and A.Ghose [33] propose an approach, named GoalBPM, to relate BPMN business process to KAOS goal models. In particular, they introduce informal and manual techniques for establishing relations among high-level stakeholder goals and business processes. These relationships are established through two steps that allow to define traceability links between goals and activities and satisfaction links between goals and processes. This method is used to support the evolution of business processes and their consistency respect to the goal models. But the purpose of this approach is quite different from our. In fact, GoalBPM can be used for verify the satisfaction of a process model against a goal model when goal changes occurs. Whilst, our approach based on GoalSpec transforms a BPMN process model into goals that can be easily interpreted by a workflow engine able to satisfy these goals. The evolution of the business processes results on new goals to be managed by the systems.

In [34], the authors propose an approach to business process management based on BDI agent technology to realize agile processes (i.e. flexible and able to proactively adapt themselves). They start with business processes expressed using GO-BPMN [35] modeling language. Differently from BPMN, in GO-BPMN workflows are attached to a goal they fulfill. Thus, this model is directly mapped into BDI agent with goals, plans and beliefs. Similarly to our idea, the authors think that the agent technology can provide an agile process execution. But what we want to realize is a workflow engine, in which agents are aware of *what* they can do, but it is not established in any way *how* to do it. They are able to find out how to complete their business process activities adapting to the available resources. We do not create a static link between the declarative level and the procedural one. By decoupling business goals and their implementation using Goalspec and adopting the workflow engine architecture shown in Fig.2, we are able to create a dynamic binding between goals and plans to reach them which are composed at run-time. Moreover, this allow us to inject new goals in the system without changing the implementation level. In addition, our approach based on a standard notation (i.e. BPMN) to model business processes does not require furthers expertise to be owned by business analysts.

Many other recent works [4,6,36,37] face with self-adaptive workflow engines. The objective of self-adaptive workflows is to make the enactment engine able to recognize anomalous situation that are not included in the specification. A promising approach in literature is to incorporate multiple strategies into the system design and to let the system to select the appropriate one that address the desired goal [7,38]. Indeed, a goal may be generally addressed in many alternative ways, each with different trade-offs. A representative approach [38] is that of modeling goals in a hierarchy that describes the expected outcome of the system.This goal model is created at design time and then each goal is instructed with the necessary implementing code which execution addresses the target goal.

Surely, these approaches should obtain more precise results but they are less flexible than our. Thus, we accept a small loss of precision in order to achieve greater flexibility and dynamism.

Final Remarks. *GoalSpec* wants to be a step toward the definition of an agent framework able to implement an adaptive workflow enactor in which goals may evolve because the user requirements are changed. Self-awareness is another important issue we are addressing. We are working to realize a kind of agent that is able to decide its own behavior with respect to evolving goals.

6 Acknowledgements

This work has been partially funded by the Innovative Document Sharing (IDS) Project funded by the Autonomous Region of Sicily (PO FESR Sicilia 2007-2013).

References

1. BPMN, O.: Business process model and notation (bpmn). www.omg.org/spec/BPMN/2.0/ (2009)
2. Van der Aalst, W., Basten, T., Verbeek, H., Verkoulen, P., Voorhoeve, M.: Adaptive workflow. Enterprise Information Systems. Kluwer Academic Publishers (1999)
3. Casati, F., Ceri, S., Pernici, B., Pozzi, G.: Workflow evolution. Data & Knowledge Engineering **24**(3) (1998) 211–238
4. Kammer, P., Bolcer, G., Taylor, R., Hitomi, A., Bergman, M.: Techniques for supporting dynamic and adaptive workflow. Computer Supported Cooperative Work (CSCW) **9**(3) (2000) 269–292
5. Serral, E., Sabatucci, L., Leonardi, C., Valderas, P., Susi, A., Zancanaro, M., Pelechano, V.: Applying a methodology for developing ami systems: the nursing home case study. In: Proceedings of the 20th International Conference on Information Systems Development Cutting edge research on Information Systems. (2011)

6. Buhler, P., Vidal, J.: Towards adaptive workflow enactment using multiagent systems. Information Technology and Management **6**(1) (2005) 61–87
7. Morandini, M., Penserini, L., Perini, A.: Towards goal-oriented development of self-adaptive systems. Proceedings of the 2008 international workshop on Software engineering for adaptive and self-managing systems (2008) 9–16
8. Winikoff, M., Padgham, L., Harland, J., Thangarajah, J.: Declarative and procedural goals in intelligent agent systems. In: International Conference on Principles of Knowledge Representation and Reasoning, Morgan Kaufman (2002)
9. Bresciani, P., Perini, A., Giorgini, P., Giunchiglia, F., Mylopoulos, J.: Tropos: An agent-oriented software development methodology. Autonomous Agents and Multi-Agent Systems **8**(3) (2004) 203–236
10. Van Lamsweerde, A.: Goal-oriented requirements engineering: A guided tour. In: Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on, IEEE (2001) 249–262
11. Yu, E., Mylopoulos, J.: Why goal-oriented requirements engineering. In: Proceedings of the 4th International Workshop on Requirements Engineering: Foundations of Software Quality. (1998) 15–22
12. Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., et al.: Software engineering for self-adaptive systems: A research roadmap. Software Engineering for Self-Adaptive Systems (2009) 1–26
13. Whittle, J., Sawyer, P., Bencomo, N., Cheng, B.H.C., Bruel, J.M.: RELAX: a language to address uncertainty in self-adaptive systems requirement. Requirements Engineering **15**(2) (March 2010) 177–196
14. Van Dyke Parunak, H., Brueckner, S.: Entropy and self-organization in multi-agent systems. In: Proceedings of the fifth international conference on Autonomous agents, ACM (2001) 124–130
15. Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in software engineering: an introduction. Kluwer Academic Publishers (2000)
16. Team, B.: Business motivation model (bmm) specification. Technical report, Technical Report dtc/06–08–03, Object Management Group, Needham, Massachusetts, USA (2006)
17. Rolland, C., Souveyet, C., Achour, C.: Guiding goal modeling using scenarios. Software Engineering, IEEE Transactions on **24**(12) (1998) 1055–1071
18. Yu, E.: Modelling strategic relationships for process reengineering. Social Modeling for Requirements Engineering **11** (2011)
19. Fuxman, A., Pistore, M., Mylopoulos, J., Traverso, P.: Model checking early requirements specifications in tropos. In: Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on, IEEE (2001) 174–181
20. Yu, L.: From requirements to architectural design–using goals and scenarios. University of Toronto.¡ http://www. cs. toronto. edu/km/GRL/fromr2a/fromr2a/straw01. pdf (2001)
21. Dardenne, A., Van Lamsweerde, A., Fickas, S.: Goal-directed requirements acquisition. Science of computer programming **20**(1) (1993) 3–50
22. Hindriks, K.V., De Boer, F.S., Van Der Hoek, W., Meyer, J.J.C.: Agent programming with declarative goals. In: Intelligent Agents VII Agent Theories Architectures and Languages. Springer (2001) 228–243
23. Howden, N., Rönnquist, R., Hodgson, A., Lucas, A.: Jack intelligent agents-summary of an agent infrastructure. In: 5th International conference on autonomous agents. (2001)
24. Rao, A.: Agentspeak (l): Bdi agents speak out in a logical computable language. Agents Breaking Away (1996) 42–55
25. Bordini, R.H., Hübner, J.F.: A java-based agentspeak interpreter used with saci for multi-agent distribution over the net (2004)

26. Hindriks, K.V., De Boer, F.S., Van der Hoek, W., Meyer, J.J.C.: Agent programming in 3apl. Autonomous Agents and Multi-Agent Systems **2**(4) (1999) 357–401
27. Dastani, M., van Riemsdijk, M.B., Dignum, F., Meyer, J.J.C.: A programming language for cognitive agents goal directed 3apl. In: Programming Multi-Agent Systems. Springer (2004) 111–130
28. Braubach, L., Pokahr, A., Moldt, D., Lamersdorf, W.: Goal representation for bdi agent systems. In: Programming multi-agent systems. Springer (2005) 44–65
29. Ribino, P., Lodato, C., Lopes, S., Seidita, V., Hilaire, V., Cossentino, M.: A norm-governed holonic multi-agent system metamodel. In: 13th International Workshop on Agent-Oriented Software Engineering. (2012)
30. ISO Technical Committee TC 154: Iso 8601 international standard date and time notation. Available at http://www.iso.org/ (1998)
31. Bordini, R., Hübner, J., Wooldridge, M.: Programming multi-agent systems in AgentSpeak using Jason. Volume 8. Wiley-Interscience (2007)
32. TC OASIS: WS-BPEL - Web Services Business Process Execution Language. Available at www.oasis-open.org (2007)
33. Ghose, A.K., Koliadis, G.: Relating business process models to goal-oriented requirements models in kaos. Faculty of Informatics-Papers (2007) 573
34. Burmeister, B., Arnold, M., Copaciu, F., Rimassa, G.: Bdi-agents for agile goal-oriented business processes. In: Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track, International Foundation for Autonomous Agents and Multiagent Systems (2008) 37–44
35. Greenwood, D., Ghizzioli, R.: Goal-oriented autonomic business process modelling and execution. Multiagent System (2009)
36. Chen-Burger, Y., Stader, J.: Formal support for adaptive workflow systems in a distributed environment. Workflow Handbook 2003 (2003) 93
37. Cao, J., Yang, J., Chan, W.: Exception handling in distributed workflow systems using mobile agents. e-Business Engineering (2005)
38. Liaskos, S., Khan, S.M., Litoiu, M., Daoud Jungblut, M., Rogozhkin, V., Mylopoulos, J.: Behavioral adaptation of information systems through goal models. Information Systems (2012)