Requirements Analysis Modeling for Buyer-Driven E-commerce Using ConGolog Agent-Oriented

Afaf B. Al-Kaddo
College of Education for Women,
Dept. of Computer Science, University of Baghdad
Afafbad@yahoo.com

Abstract
This paper describes an agent-oriented requirements engineering approach. Agent-oriented techniques represent an exciting new means of analyzing, designing and building complex software systems. They have the potential to significantly improve current practice in software engineering and to extend the range of applications that can feasibly be tackled. i * can be used to model social dependencies between agents and how process design choices affect the agents' goals. Agent-oriented approaches are becoming popular in software engineering, both as architectural frameworks, and as modeling frameworks for requirements engineering and design. i* is an informal diagram-based language for early-phase requirements engineering that supports the modeling of social dependencies between agents and how process design choices affect the agents' goals both functional and non-functional. ConGolog is an expressive logic-based formalism for specifying processes that involves multiple agents. The two formalisms complement each other well, and in this work, we used the combination of the two frameworks in requirements engineering. The i* SR-diagram language is extended with process specification annotations, which allow the SR model of a system to be refined and then mapped into a ConGolog model. The mapping must satisfy a set of mapping rules, which ensure that it specifies which elements in the two models are related and that the models are consistent.

Keywords: *Agent-oriented, Requirements Engineering, Buyer-Driven E-commerce, i*, ConGolog*

1. Introduction

To support the development of agent based systems, suitable software engineering methods and tools are required. So far, most efforts in this area have been directed at the design phase of software development. In this paper, we focus mainly on the requirements engineering phase. The i* framework [1] was developed for early phase requirements analysis. But i* is not a formal language and it has limited support for describing complex processes. ConGolog [2, 3] is a formal language for process specification

and agent programming. It supports the formal specification of complex multi agent systems, but lacks features for modeling the rationale behind design choices. The two frameworks complement each other well and it would be good to have a methodology for using them in combination.
Agent orientation is emerging as a new paradigm for constructing software systems. New kinds of systems are now being developed based on the concept of software agent. Agent orientation is offering an exciting new way of thinking about what software is and how it should be constructed. Agent orientation offers a higher level of abstraction for thinking about the characteristics and behaviors of software systems. It can be seen as part of an ongoing trend towards greater interactivity in conceptions of programming and software system construction [4, 5, and 6]. The concept of agent can be developed to serve as the central construct in a new kind of modeling and analysis to respond to today's needs. The agent concept can be instrumental in bringing about a shift to a much richer, socially-oriented ontology that is needed to characterize and analyze today's systems and environments.

In this paper, we used an approach to the combined use of the i* and ConGolog frameworks for requirements analysis. The i* framework will be used to model different alternatives for the desired system, analyze and decompose the functions of the different actors, and model the dependency relationships between the actors and the rationale behind process design decisions. The ConGolog framework will be used to formally specify the system behavior described informally in the i* model. the i* technique [7, 8], which allows a better description of the organizational relationships among the various agents of a system as well as an understanding of the rationale of the decisions taken. The ConGolog model will provide more detailed information about the actors, tasks, processes, and goals in the system, and the relationships between them. To reduce the gap between i* and ConGolog models, notation involving the use of process specification annotations in i* SR diagrams will be introduced. A set of mapping rules to map the elements of the annotated SR diagram to appropriate ConGolog entities and ensures that the two models are consistent will be used. Finally, a method for combination both the i* and ConGolog frameworks are used for early to late phase requirements engineering.

2. The i* Modeling Framework

The i* framework [1, 9] was developed for modeling and analyzing organizations to help support business process reengineering and requirements engineering. The framework focuses on modeling intentional and strategic relationships between actors. It consists of two models — the

Strategic Dependency (SD) model (represents the intention level) and the Strategic Rationale (SR) model (represents the rational level). The SD model is used to capture the network of relationships that hold among the actors. The SR model describes in more detail the alternative methods that the actors have for accomplishing their goals and tasks. It helps the modeler in understanding the existing processes, and in generating alternatives in order to reach a new process design that better addresses the organization's objectives.

SD model is a graph consisting of a set of actors (nodes) and links among the nodes. Each node represents an actor, and each link between the actors represents how one actor depends on another for something in order to accomplish a goal or task. The depending actor is called the depender, the actor who is depended upon by another is called the dependee, and the object on which the dependency relationship centers are called the dependum (can be a resource, task, goal or softgoal). An actor can be an agent, a role, or a position. An agent is a concrete, physical actor or subsystem. A role is a function in an organization. A position is a set of roles played by one agent (actor) that is institutionalized in an organization. For example, in a buyer-driven e-commerce process, the Customer As Buyer (a role) depends on a Middleman As Seller agent to organize purchasing. The Customer As Buyer is a depender, the Middleman As Seller is a dependee, and the goal "Service Be Purchased" is the dependum in a dependency relationship that the Customer As Buyer has on the buyer-driven e-commerce.

We can distinguish four types of dependencies: goal-, task-, resource-, and softgoal-dependencies. In a goal-dependency, the depender (agent) depends on the dependee to bring about some desired state. The dependee is free to decide how to achieve the goal on his own. A task-dependency specifies that the depender (agent) depends on the dependee to complete some activities (i.e., how the task is to be performed, but not what and why). A resource-dependency specifies that the depender (agent) depends on the dependee for the availability of an information or physical resource (entity). The soft-goal dependency is similar to the goal dependency, except that the condition is not precisely defined at the start of the process, i.e., the goals in a sense involves subjective aspects, that gradually are clarified during the development process. A softgoal-dependency expresses that the depender depends on the dependee to achieve a softgoal, i.e. a goal that can be achieved to varying degrees, and needs to be optimized. In a dependency, the depender may be vulnerable in case that dependee fails to bring about the dependum. There are three degrees of strength of dependencies in the SD model: open (uncommitted), committed, and critical.

The Strategic Dependency model describes the network of intentional relationships among actors. Actors depend on each other for goals to be achieved, tasks to be performed, and resources to be furnished. These dependencies are intentional in that they are based on underlying concepts such as goal, ability, commitment, belief, and so on [10]. The Strategic Rationale (SR) model of i* model addresses this.

The Strategic Rationale model describes and supports the reasoning that each actor has about its relationships with other actors, its alternative means to achieve its goals, and how the qualitative expectations of actors are satisfied by these alternatives. Actors in i* are strategic in that they evaluate their social relationships in terms of opportunities that they offer, and vulnerabilities that they may bring. Strategic actors seek to protect or further their interests.

The SR model is a graph consisting of four types of nodes (present in the SD model): goal, task, resource, and softgoal nodes, and two types of links, means-ends links and task-decomposition links. Task-decomposition links describe how a task can be decomposed into subtasks or subgoals. Means-ends links specify how a goal may be achieved. They provide information about why an actor would perform a task, pursue a goal, need a resource, or want a softgoal. From the softgoals, the modeler can tell why one alternative may be chosen over others.

2.1. The Strategic Dependency (SD) Model

Figure.1 shows a Strategic Dependency (SD) model for a buyer-driven ecommerce system. In this system, there are 3 actor nodes (customer as buyer, middleman as seller and supplier); the customer depends on a middleman to find a service provider who is willing to accept a price set by the customer. The customer submits a priced request to a middleman. The middleman forwards the request to suppliers. If a supplier decides to accept the request, it makes an agreement with the middleman. The middleman expects the customer to pay for the purchase in time.

In this system, each link between these agents/roles represents how one agent/role depends on another for something, for example, the supplier depends on the middleman to attract more customers, which in turn relies on the middleman having loyal customers. The customer depends on the supplier for quality service (since the service is coming from the supplier, not from the middleman). Finally, the scheme works only if the prices set by customers are acceptable to suppliers.

The Strategic Dependency model distinguishes among several types of dependencies based on the degree of freedom the dependee has and the ontological category of the dependum when delivering the dependum to the depender. In a goal dependency, an actor depends on another to make a condition in the world come true. In Figure1, the goal dependency Low

Price Service Provider is Found from the customer to the middleman means that it is up to the middleman to decide how to find the low price service provider.

In a task dependency, an actor depends on another to perform an activity. The activity description specifies a particular course of action. For example, the task dependency Name a Price [Service] expresses that the customer depends on the middleman to name his own price for the service in need by specifying the standard procedure for naming a price.

In a resource dependency, an actor depends on another for the availability of an entity. The depender takes the availability of the resource to be unproblematic. In Figure1, the customer's dependency on the supplier for agreement on price is modeled as a resource dependency.

The fourth type of dependency, softgoal dependency, is a variant of the first. It is different from a (hard) goal dependency in that there are no a priori, cut-and-dry criteria for what constitutes meeting the goal. The meaning of a softgoal is specified in terms of the methods that are chosen in the course of pursuing the goal. The dependee contributes to the identification of alternatives, but the depender makes the decision. The notion of softgoal allows the model to deal with many of the usually informal concepts. For example, the customer's dependency on the supplier for good quality service can be achieved in different ways. The desired degree of how good the quality should be is ultimately decided by the depender.

2.2. The Strategic Rational (SR) Model

The Strategic Rationale (SR) model provides a more detailed level of modeling by looking "inside" actors to model internal intentional relationships. Intentional elements (tasks, resources, goals, and softgoals) in SR models appear as external dependencies and as internal elements linked by means-ends, task decompositions and contribution relationships.

Figure 2 is an SR model of the buyer-driven e-commerce system with middleman corresponding to Figure 1. In this model, the internal rationale of the customer and the middleman are elaborated. The customer's main goal is that Service Be Purchased [Service], for the customer as buyer this is an internal goal. To achieve this goal, the internal task Purchase By Naming My Own Price [Service] must be accomplished.

This internal task is connected to the internal goal with a means-ends link, but this task has two sub-elements connected to it through decomposition links – the sub-task Name A Price [Service], and the sub-goal Low Price Service Provider Be Found. The purchase task is achievable only if all its sub-elements are accomplished. Naming one's own price contributes positively (+) to the buyer's softgoal of Low Price, but negatively (-) to Flexibility [Purchasing] because preferences about schedule, choice of airline, *etc.*, could not be accommodated.

2.2. The Strategic Rational (SR) Model

The Strategic Rationale (SR) model provides a more detailed level of modeling by looking "inside" actors to model internal intentional relationships. Intentional elements (tasks, resources, goals, and softgoals) in SR models appear as external dependencies and as internal elements linked by means-ends, task decompositions and contribution relationships. Figure 2 is an SR model of the buyer-driven e-commerce system with middleman corresponding to Figure 1. In this model, the internal rationale of the customer and the middleman are elaborated. The customer's main goal is that Service Be Purchased [Service], for the customer as buyer this is an internal goal.

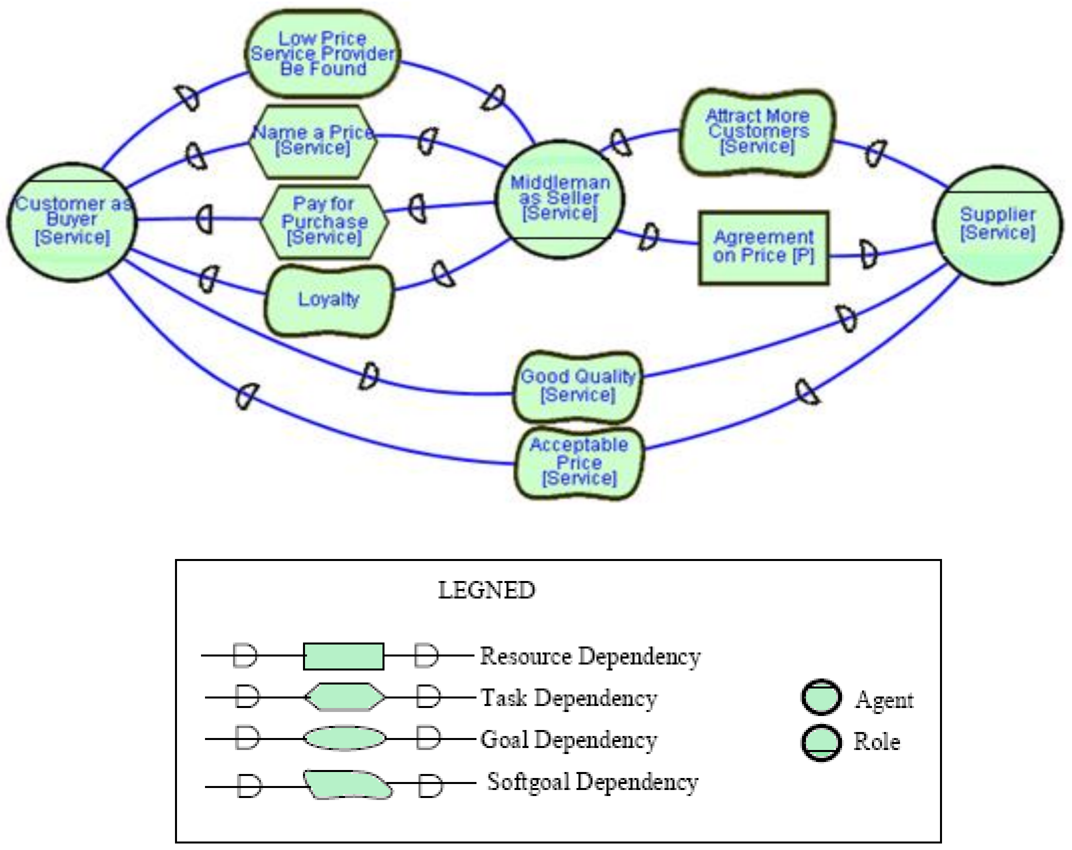

Low Price Service Provider Be Found
Name a Price [Service]
Pay for Purchase [Service]
Loyalty
Customer as Buyer [Service]
Middleman as Seller [Service]
Attract More Customers [Service]
Agreement on Price [P]
Supplier [Service]
Good Quality [Service]
Acceptable Price [Service]
LEGNED
Resource Dependency
Task Dependency
Goal Dependency
Softgoal Dependency
Agent
Role

Figure 1. Strategic Dependency Model of a Buyer-driven e-commerce System with Middleman

To achieve this goal, the internal task Purchase By Naming My Own Price [Service] must be accomplished. This internal task is connected to the internal goal with a means-ends link, but this task has two sub-elements connected to it through decomposition links – the sub-task Name A Price [Service], and the sub-goal Low Price Service Provider Be Found. The purchase task is achievable only if all its sub-elements are accomplished.
Naming one's own price contributes positively (+) to the buyer's softgoal of Low Price, but negatively (-) to Flexibility [Purchasing] because preferences about schedule, choice of airline, etc., could not be accommodated.

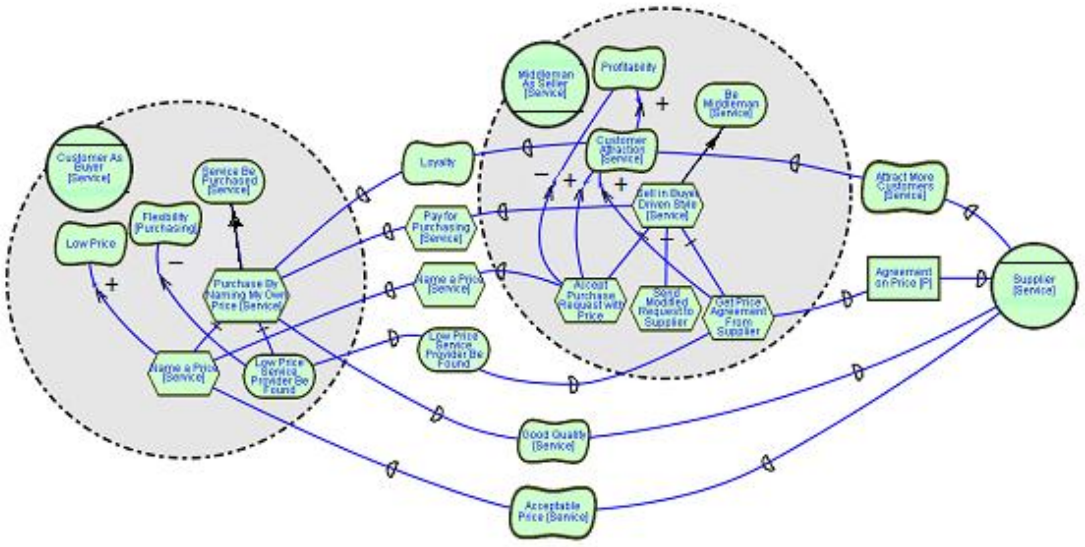

Customer As Buyer [Service]
Service Be Purchased [Service]
Flexibility [Purchasing]
Low Price
Purchase By Naming My Own Price [Service]
Name a Price [Service]
Low Price Service Provider Be Found
Loyalty
Pay for Purchasing [Service]
Name a Price [Service]
Low Price Service Provider Be Found
Middleman As Seller [Service]
Profitability
Be Middleman [Service]
Customer Attraction [Service]
Sell in Buyer Driven Style [Service]
Accept Purchase Request with Price
Send Modified Request to Supplier
Get Price Agreement From Supplier
Attract More Customers [Service]
Agreement on Price [P]
Supplier [Service]
Good Quality [Service]
Acceptable Price [Service]
+
−
+
−
+
+
+
−

Figure 2. Strategic Rationale Model of a Buyer-driven e-commerce System with Middleman

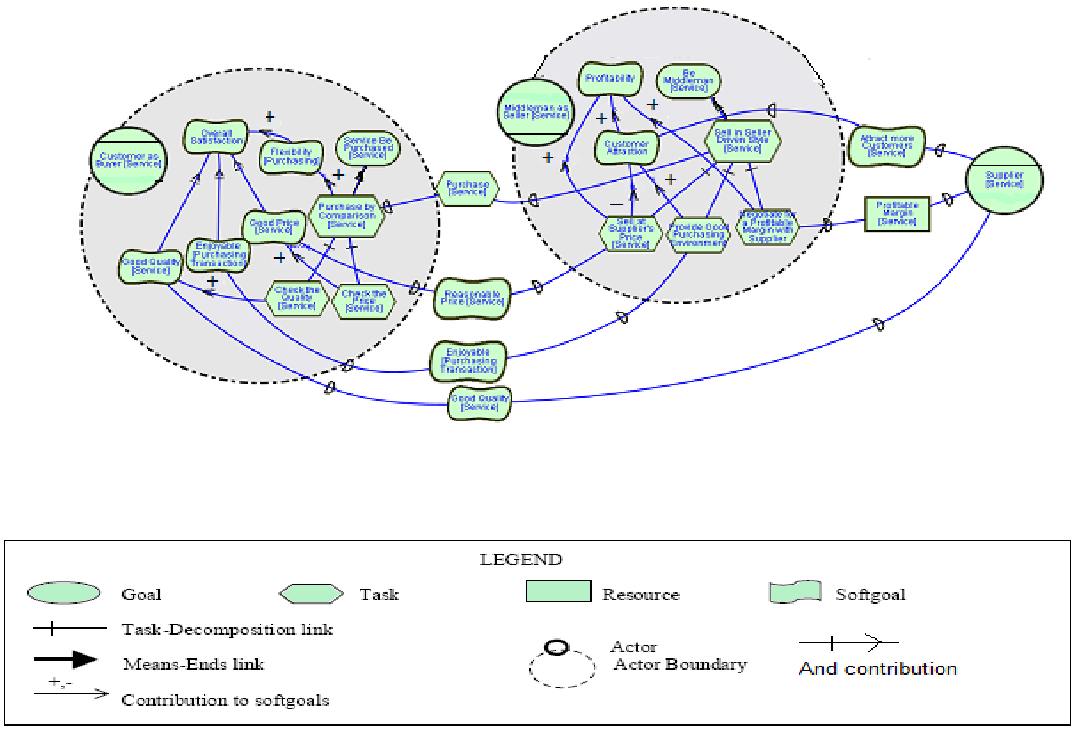

Customer as Buyer [Service]
Overall Satisfaction
Flexibility [Purchasing]
Service Be Purchased [Service]
Good Price [Service]
Purchase by Comparison [Service]
Good Quality [Service]
Enjoyable Purchasing Transaction]
Check the Quality [Service]
Check the Price [Service]
Purchase [Service]
Reasonable Price [Service]
Enjoyable [Purchasing Transaction]
Good Quality [Service]
Middleman as Seller [Service]
Profitability
Be Middleman [Service]
Customer Attraction
Sell in Seller Driven Style [Service]
Sell at Supplier's Price [Service]
Provide Good Purchasing Environment
Negotiate for a Profitable Margin with Supplier
Attract more Customers [Service]
Profitable Margin [Service]
Supplier [Service]
LEGEND
Goal
Task
Resource
Softgoal
Task-Decomposition link
Means-Ends link
Contribution to softgoals
Actor
Actor Boundary
And contribution

Figure 3. Strategic Rationale Model of a Seller-driven e-commerce System with Middleman

Figure 3 shows the constructed of buyer-driven system. The Be Middleman goal is addressed by an alternate task Provide Comparison Shopping Service. By doing this, the middleman is more profitable as he could earn more revenue by maximizing the margin. The higher price might impact Customer Attraction negatively, the middleman could regain some customer attraction by Provide Good Purchasing Environment and the customer can enjoy more Flexibility [Purchasing] at the same time and get overall satisfaction.

3. ConGolog Modeling Framework

ConGolog [2, 3] is a framework for process modeling and agent programming, it is also a language for high-level programming. ConGolog is an extension of the situation calculus for modeling dynamic worlds and supports complex actions as well as a logic programming language for agents. The ConGolog framework can be used to model *complex processes* involving nondeterminism, loops, multiple agents, and concurrency. Because it is logic-based, the framework can accommodate incompletely specified models, either in the sense that the initial state of the system is not completely specified, or that the processes involved are nondeterministic and may evolve in any number of ways.

A ConGolog model of a domain includes two components. The *specification* of the domain dynamics (how to model the state, what is the initial state of the domain, what actions can be performed, when the actions can be performed and what their effects are). The *specification of the process of interest* (behavior of the agents involved in the domain). A dynamic domain in ConGolog and the situation calculus can be modeled using the following entities:

• *Actions:* are functions denoting individual instantaneous events that change the state of the world.

• *Agents*: are agents to be modeled in the system; these are represented by constants, *e.g.*, `bd1`, the customer as buyer.

• *Situations*: are histories of the actions that have occurred, viewed as sequences of actions. The initial situation S0(no actions have yet occurred) and the successive situations built up using binary function `do(B,S)`, denotes the situation which results from action `B` being performed in situation `S`. The sequence of actions in a history, and the order in which they occur, are obtained from a situation term. For example, `do(B3,do(B2,do(B1,s0)))` represents the history of the sequence of actions `B1`, `B2`, and then `B3` are performed starting in the initial situation `s0`.

• *Primitive actions*: all changes to the world are taken to be the result of named primitive actions that are performed by some agent in the system; primitive actions are represented by terms.

• *Fluents*: these are predicates or functions representing relations, properties, or functions whose value may change from situation to another and they are represented by symbols that take a situation term as their last argument.

The dynamics of a domain are specified using three kinds of axioms:

- *Action precondition axioms*: these axioms state the conditions under which an action can be enabled(what preconditions must hold in order an action to be executable in a situation); these use the predicate poss(B,S), which represents that action B is possible(executable) in situation S. For example, in buyer-driven e-commerce domain, we have the following precondition axiom:

poss(acceptAgreementReq(Middleman,BD,ReqID,Price),S)≡
agreementReqRcvd(Middleman,BD,ReqID,Price,S)∧
LowPrice(Middleman,Price,S)

This says that in situation `S`, `Middleman` may perform the action of accepting a request `ReqID` from `BD` to sell on Price if and only if he has received a request to that effect and the price is OK for him.

- *Successor state axioms*: these axioms specify how the fluents are affected by the actions in the domain. For example, in buyer-driven e-commerce domain, we have the following successor axiom:

```
agreementReqRcvd(Middleman,BD,ReqID,Price,do(A,S))
                          ≡
    A = requestAgreement(BD,Middleman,Price) ∧
                    reqCtr(S) =
                      ReqID ∨
  agreementReqRcvd(Middleman,BD,ReqID,Price,S)
```

This says that `Middleman` has received a request `ReqID` from `BD` to agree to sell on Price in situation `do(A,S)` if and only if the action `A` is such a request and the value of the request counter is `ReqID` or if he had already received such a request in situation `S`.
Successor state axioms can be generated automatically from a specification of the effects of primitive actions if we assume that they specify all of the ways that the value of the fluent may change. Successor state axioms were introduced in [11] and provide a solution to the frame problem. [3] describes a convenient high-level notation for specifying the effects (and preconditions) of actions and a tool that compiles such specifications into successor state axioms.

- *Initial situation axioms*: these axioms specify the initial state of the modeled system. For example:

customerPrice(kk,s0)=[200], this says that agent kk has a price 200 in the initial situation.

4. Combination Use of i* and ConGolog[12]

In i* SR diagram many aspects of processes are represented, but it is imprecise and produce incomplete models. For instance, it is not specified if the subtask in a task decomposition link is repeated or not (*i.e.*, performed once or more). In a ConGolog model, the process must be precisely and completely specified (non-deterministic processes are allowed). We need to fill the gap between SR diagram and ConGolog model. We will use a set of annotations to SR diagrams that allow the missing information to be specified. We want to have a mapping between the annotated SR (ASR) diagram and the ConGolog model to specify which parts are related of each model. This allows us identify which parts of the ConGolog model need to be changed when the SR model is modified and vice versa. We define a set of mapping rules that define what mappings are allowed to respect the semantics of both frameworks.

4.1. Annotations of SR Diagram

Two types of annotations are defined: composition annotations and link annotations. Composition annotations are applied to groups of decomposition links in the SR model. These annotations explain how the linked subtasks/subgoals are to be composed in order to perform the super task/goal.

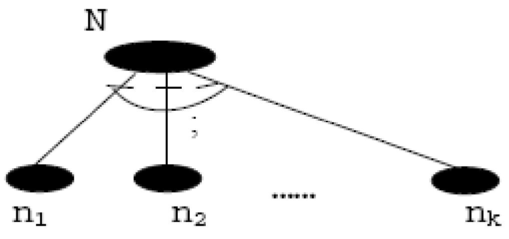


Figure 4. Sequence Annotation Applied to a Group of Decomposition Links

There are four types of composition annotations: sequence “;”, alternative (nondeterministic branch) “|”, concurrency execution “||”, and concurrency with different priority “>>”. For example, in Figure 4, if we put sequence annotation “;” on the links as the subtasks/subgoals are to be sequentially performed left to right to complete the decomposed task.
Link annotations are applied to individual decomposition links connecting a task/goal with the linked subtask/subgoal. Link annotations describe what condition the subtask/subgoal is to be performed and whether it should be perform once or more. There are five types of link annotations: the while-loop annotation “*while(condition)”, the for-loop annotation “*for(variable,listOfValues)”, the interrupt annotation “*whenever(variableList,condition)”, the if annotation “if(condition)”, and the pick annotation “pick(variableList,condition)”. The first three are for iterating the subtask/subgoal, and the last is for non-deterministic picking values for the variables in the subtask that satisfy the condition. All of the annotations correspond to ConGolog operators and are taken to have the same meaning.

4.2. Mapping Rules

We must define a mapping m from the annotated SR (ASR) diagram elements to the entities in the ConGolog model. We define mapping rules that respect the rules, which arise from the semantics of the two formalisms to ensure consistency between the ASR model and the ConGolog model. There are two sets of mapping rules:

4.2.1. Mapping Rules for SR Diagram Nodes

We have to define mapping rules for each type of nodes in the ASR model (i.e. agent, goal, task, role, and position nodes). These ensure that the nodes are mapped into appropriate ConGolog entities.

- Rule for **agent nodes** is:

*If **gn** is an agent node, b(gn)=<ag, ag_behavior>, where **ag** is a ConGolog agent and **ag_behavior** is a ConGolog procedure representing the behavior of the agent.*

We use **b_agent(gn)** to refer to the agent **ag** and **b_behavior(gn)** to refer to the agent's behavior **ag_behavior**.

- Rule for **role/position nodes** is:

*If **gn** is a role/positiont node, b(gn)=gn_behavior, where **gn_behavior** is a ConGolog procedure representing the behavior associated with the role/position.*

- Rule for **task nodes** is:

*If **gn** is a task node, b(gn)=t, where **t** is a ConGolog procedure or primitive action representing the task.*

- Rule for **goal nodes** is:

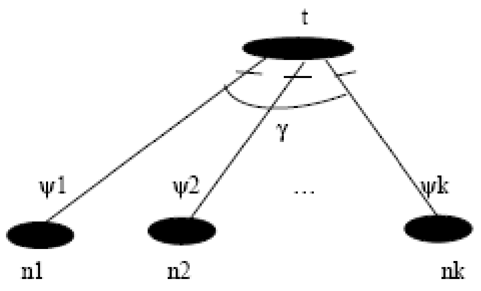
*If **gn** is a goal node, b(gn)=<g, achieve_g>, where **g** is a ConGolog fluent (primitive or defined) and achieve_g is a ConGolog procedure specifying a selected set of means for achieving the goal which the agent will perform when he adopts it.*

We use b_fluent(gn) to refer to the fluent **g** and b_achieve(gn) to refer to the procedure achieve_g.

4.2.2. Mapping Rules for SR Diagram Links

The two types of links in the ASR model, task decomposition links and means-ends links have mapping rules. The task decomposition links' mapping rule is as follows:

*If **t** is a task node that is decomposed into nodes n1,..,nk by task decomposition links, where a composition annotation* **g** *is applied to the* *group of decomposition links and link annotations* **yi** *are applied to the individual decomposition link between* **t** *and* ***ni*** *, i.e.,*



then m(t) must be a ConGolog procedure of the form:

```
proc(t(parameterList),
ψ1′(m_proc(n1))
γ′
ψ2′(m_proc(n2))
γ′
...
γ’
ψk′(m_proc(nk))
)./*end of procedure*/
```

where γ ′(ψi ′) is the ConGolog operator corresponding to the annotation γ (ψi) and m_proc(ni) is the ConGolog procedure associated with node ni, i.e. m(ni) if ni is a task node and m_achieve(ni) if ni is a goal node. Thus, the body of the ConGolog procedure associated to a task node **t** must be exactly as specified by the decomposition defined for **t** in the ASR diagram. It should be straightforward to automate the translation. The mapping rule for means-ends links (goal decomposition links) is very similar to the one above. Essentially, it requires that the b_achieve(bg) procedure associated with a goal node **bg** be as specified by the decomposition defined for b**g** in the ASR diagram. There is one additional requirement: that the fluent associated with the goal b_fluent(bg) be true when the procedure completes execution.

5. Operationalization

The dependencies between agents, roles and positions in the SR model indicate that the depender depends on the dependee to accomplish one of his tasks or goals or to supply some resource. The *i** SR model abstracts all details of the associated interactions between the agents (requests, communication/interaction protocols). The ConGolog model focuses on the operational aspects of processes rather than their strategic/social aspects. Dependency relationships themselves are not represented in the ConGolog model. Instead, the modeler operationalizes the dependencies in the ASR model. After this, the tasks and goals that result are mapped into ConGolog. The details of how a dependency is operationalized depend on the particulars of the case and must be filled in by the modeler.

5.1. A Method to Combine Use of i* and ConGolog Frameworks

1). Building the Strategic Dependency (SD) Model for the Application.
The SD model specifies the agents, roles, positions, and the intentional dependency relationships between them for the application.

2). Building the Strategic Rationale (SR) Model for the Application.
The SR model identifies the tasks/goals/softgoals inside the agents, roles, and positions, the decompositions of the tasks/goals, and the contributions of the various alternatives to the softgoals. Dependency relationships are specified between nodes inside the agents/roles/positions.

3). Building the Annotated Strategic Rationale (ASR) model for the application.
This involves the following substeps:
• Reduce unnecessary information from the SR model. Softgoals and their links and dependencies are suppressed.
• Goals that cannot always be achieved are weakened/conditionalized.
• Operationalizing the dependencies: Specified the interactions through which the task/goal/resource dependencies are realized by the actors.
• Using annotations to fill out process details and to specify how goals and tasks are to be composed. Every task/goal will be decomposed into atomic subtasks/subgoals if applicable.

4). Developing the Initial ConGolog Model
Maps elements in the ASR model into entities in the ConGolog model while conforming to the mapping rules; complete the initial ConGolog model by specifying the fluents, primitive actions, precondition axioms, and successor state axioms for the system.

Repeat steps 1 to 4 until objectives are met.

Whenever the *i** model or ConGolog model has to be modified based on the results of the validation step, the modeler refines the corresponding part of the other model. This continues until the objectives are accomplished.

5). Producing the Requirements Specification Document.
Steps 2, 3, and 4 could be performed in sequence, but need not be. Instead, work on different tasks/goals, decomposing/refining them, and introducing annotations and ConGolog domain specifications.

6. Buyer Driven E-commerce Process

6.1. Building SD Model

A strategic Dependency (SD) model was shown in Figure 1. The SD model specifies the dependencies that actors have on each other to make the modeler understanding of the "why" behind the process.

6.2. Building SR Model

The SR model of Buyer-driven ecommerce was shown in Figures 2 and 3. The SR model shows more detailed level of modeling is performed by looking "inside" the actors to model internal relationships.

6.3 Refinement of Annotated SR Model

Annotated SR diagram is a more detailed description of the process and that can be mapped into a ConGolog model. Figure 5 shows the buyer part because the diagram is large. This diagram is developed in stages: **Simplify the SR model by suppressing unnecessary information from the SR model**, we simplify the SR diagram by suppressing softgoal nodes and their links with other nodes from the initial SR model because these are not relevant to mapping the SR model into a ConGolog model. **Relativizing goals that cannot Always be accomplished,** In Figure 3, the goal “Service Be Purchased [Service]” is accomplished only when prices set by customers are acceptable to suppliers. To allow for failure, this will weaken the goal to “Service Benn Purchased [Service]”, which is accomplished once the supplier with low price is found. The scheme of achieving this goal are refined into “try obtain agreement on price [P]” and “try to find low price provider” in the customer and “try to find agreement price[P]” in the Middleman. **Using annotations to fill out process details,** annotations are added to specify how subtasks are composed and whether they are iterated or conditional Figure 5.

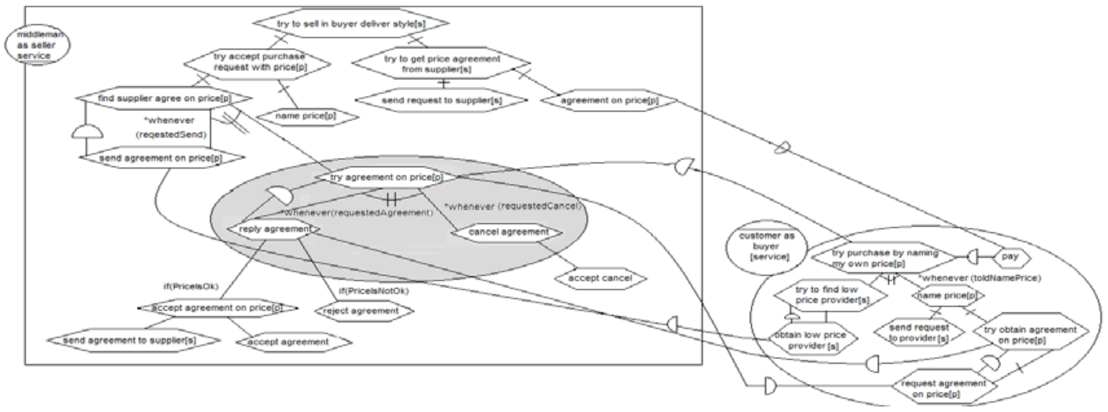
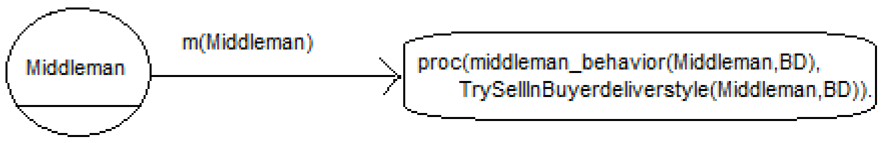

middleman as seller service
try to sell in buyer deliver style[s]
try accept purchase request with price[p]
try to get price agreement from supplier[s]
find supplier agree on price[p]
name price[p]
send request to supplier[s]
agreement on price[p]
*whenever (reqestedSend)
send agreement on price[p]
try agreement on price[p]
*whenever(requestedAgreement)
*whenever (requestedCancel)
reply agreement
cancel agreement
accept cancel
if(PriceIsOk)
if(PriceIsNotOk)
accept agreement on price[p]
reject agreement
send agreement to supplier[s]
accept agreement
customer as buyer [service]
try purchase by naming my own price[p]
pay
*whenever (toldNamePrice)
try to find low price provider[s]
name price[p]
obtain low price provider [s]
send request to provider [s]
try obtain agreement on price[p]
request agreement on price[p]

Figure 5. The Annotated SR Model for the Middleman

For example, in the shadowed area of Figure 4, the task node “try agree on price[p]” is decomposed into two subtask nodes “reply agree on price[p]” and “cancel agree on price[p]”, and the group of decomposition links is labeled with the concurrency annotation “||”, meaning that the two subtasks will be performed concurrently. The link connecting the node “reply agreement” with “try agree on price[p]” is accompanied by the annotation “*whenever(requestedAgreement)”, which means that whenever there is a request for agreement from the customer, the task “reply agreement” will be performed.

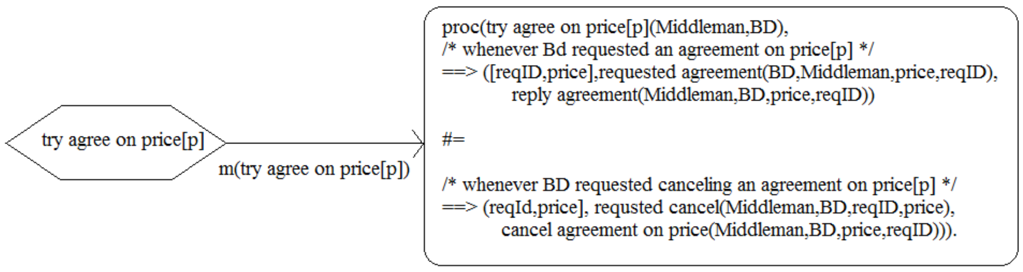
6.4. Developing the ConGolog Model

We have to map entities in the ASR diagram into corresponding elements of a ConGolog model. Bellow some examples explaining the mapping rules to obtain the initial ConGolog model. First, all nodes must be mapped. For example, the behavior of the role node “Middleman” is mapped into a ConGolog procedure as follows:



```
proc(middleman_behavior(Middleman,BD),
      TrySellInBuyerdeliverstyle(Middleman,BD)).
```

The second example the task node "try agree on price[p]" in the shadowed area of Figure.4, is mapped into ConGolog procedure as follows:



```
proc(try agree on price[p](Middleman,BD),
/* whenever Bd requested an agreement on price[p] */
==> ([reqID,price],requested agreement(BD,Middleman,price,reqID),
        reply agreement(Middleman,BD,price,reqID))

#=

/* whenever BD requested canceling an agreement on price[p] */
==> (reqId,price], requsted cancel(Middleman,BD,reqID,price),
       cancel agreement on price(Middleman,BD,price,reqID))).
```

All links must be mapped. The task node “try agree on price” is decomposed into subtasks “reply agreement” and “cancel agreement on price” these subtasks are performed concurrently whenever an associated condition holds.

After mapping, we must specifying the domain dynamics. Primitive actions and fluents are used to model the features of the domain. Successor state axioms will be used to model how the primitive actions affect the fluents. The ConGolog domain specification includes:

- A primitive action acceptAgreementReq(Middleman,BD,ReqID,Price), that models the action of the Middleman to accept the request ReqID from the BD to sell on price[p].

- An exogenous action occupyDateFromMiddleman(Middleman,Price), that models an action by an agent outside the system which causes the Middleman to occupy Price.

- A predicate fluent occupyAcknowledged(Middleman,Price), that models the fact that the occupation of the Priceon Middleman’s price has been acknowledged (in response to the exogenous action described above).

- A functional fluent MiddlemanPrice(Middleman), whose value is the price**.**

- A defined fluent waitingForAgreementAnswer(BD,Middleman,Price,Selling), that models the fact that the BD is waiting for the Middleman to agree to have the Selling on Price.

A successor state axiom for each primitive are specified. For example, the successor axiom for the fluent occupyAcknowledged(Middleman,Price) is as follows:

```
holds(occupyAcknowledged(Middleman,Price),do(A,S)):-
A = acknowledgeOccupy(Middleman,Price);
holds(occupyAcknowledged(Middleman,Price), S).
```

This says that a request to occupy a Price has been acknowledged in the situation that is the result of doing action A in situation S if and only if action A is to acknowledge this occupation, or if the occupation has already been acknowledged in situation S.

We specify an action precondition axiom for each primitive action. For example, the precondition axiom for the action acceptAgreementReq(Middleman,BD,ReqID,Price) is as follows:

```
poss(acceptAgreementReq(Middleman,BD,ReqID,Price),S):-
holds(agreementReqRcvd(Middleman,BD,ReqID,Price),S),
holds(dateFree(Middleman,Price),S).
```

This says that the action can be performed in situation S if Middleman has received a request ReqID from BD to sell on Price

6.5. Simulation and Validating of the ConGolog Model

A complete ConGolog models can be executed to run process simulation experiments. To do this, the modeler must first specify an instance of the overall system. We do this by defining a main procedure, for example:

```
proc(main,[customer_behavior(init1,bd1 )#
middleman_behavior(bd1,init1) #
supplier_behavior(agn,bd1) # ]).
```

Here, there is a customer agent init1, a middleman bd1, and a supplier agn. # is the concurrent execution operator in the ConGolog implementation.

6.6. Producing the Requirements Document by Refining the i* and ConGolog Models

The above steps will be repeated if inadequacies or errors are found in the modeled. If after specifying the ConGolog model and performing simulation, we determine that the *i** model lacks some element of the desired requirements, so, it will be modified appropriately. Similarly if we find that the ConGolog model needs an additional details or aspects of the *i** model, modifications will be made to it. A requirement specification document is produced, once a satisfactory model of the required system has been developed.

7. Conclusion

We used an approach to requirements engineering that integrates the two frameworks the i* and ConGolog. *i** can be used to model social dependencies between agents and ConGolog can be used to model complex processes formally. The advantages of using the two frameworks are:

■ Extended *i** SR diagrams by using a set of *process specification annotations*. By using annotated SR diagrams, we can provide a precise specification of the processes of interest, and the producing diagrams can then be mapped directly into a ConGolog model.

■ Using a set of *mapping rules* to constrain the modeler to map the annotated SR diagram into ConGolog model and ensure that the models are consistent. This allows the modeler to trace corresponding elements in the two models when changes are made.

References

[1] Yu, E. S. K., "Modelling Strategic Relationships for Process Reengineering", Ph.D. thesis, Dept. of Computer Science, University of Toronto, Toronto, ON, **(1995)**.

[2] G. De Giacomo, Y. Lespérance and H. J. Levesque, “ConGolog, a concurrent programming language based on the situation calculus”, Artificial Intelligence", vol. 121, **(2000)**, pp. 109-169.

[3] Y. Lespérance, T. G. Kelley, J. Mylopoulos and E. S. K. Yu., “Modeling Dynamic Domains with ConGolog”, Advanced Information Systems Engineering, 11th International Conference,

CAiSE-99, Proceedings, Heidelberg, Germany, LNCS, Springer-Verlag, Berlin, vol. 1626, **(1999)** June, pp. 365-380.

[4] A. Newell, “The Knowledge Level”, Artificial Intelligence, vol. 18, **(1982)**, pp. 87-127.

[5] D. G. Bobrow, “Dimensions of Interaction":AAAI-90 Presidential Address”, AI Magazine, vol. 12, no. 3, **(1991)**, pp. 64-80.

[6] G. Wagner, Y. Lesperance and E. Yu, “Agent-Oriented Information Systems 2000”, Proceedings of the 2nd International Workshop at CAiSE*00, Stockholm, iCue Publishing, Berlin. ISBN 3-8311-0093-4. See also http://aois.org, **(2000)** June.

[7] E. Yu and J. Mylopoulos. “Understanding "Why" in Software Process Modelling, Analysis, and Design”, Proceedings of the 16th International Conference on Software Engineering, 16th International Conference on Software Engineering (ICSE’94) Sorrento, Italy, **(1994)**, pp. 159-168.

[8] P. Traverso, “Specifying and analizing early requirements in Tropos”, Requirements Engineering Journal, vol. 9, no. 2, **(2004)**, pp. 132-150.

[9] E. S. K. Yu, “Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering”, Proc. of the 3rd IEEE Int. Symp. on Requirements Engineering (RE’97), Washington, DC, **(1997)**, pp. 226-235.

[10] E. Yu, “Modelling Strategic Relationships for Process Reengineering”, Ph.D. thesis, also Tech. Report DKBS-TR-94-6, Dept. of Computer Science, University of Toronto, **(1995)**.

[11] R. Reiter, “The Frame Problem in the Situation Calculus: A Simple Solution (Sometimes) and a Completeness Result for Goal

Regression”, Artificial Intelligence and Mathematical Theory of Computation: Papers in Honor of John McCarthy, V. Lifschitz, Academic Press, San Diego, CA, **(1991)**, pp. 359-380.

[12] X. Wang and Y. Lespe'rance, “Agent-Oriented Requirements Engineering Using ConGolog and i*”, Dept. of Computer Science, York University, Toronto, On, M3J 1P3, Canada, **(2002)**.

نمذجة تحليل الاحتياجات ل Buyer- Driven التجارة الإلكترونية باستخدام وكيل المنحنى ConGolog

د. عفاف بديع القدو

جامعة بغداد / كلية التربية للبنات / قسم الحاسبات

المستخلص :

تصف هذه الورقة المتطلبات الهندسية موجهة الوكيل. تمثل التقنيات موجهة الوكيل وسيلة جديدة ومثيرة من تحليل وتصميم وبناء أنظمة البرمجيات المعقدة. ولديها القدرة على تحسين الممارسة الحالية في هندسة البرمجيات وتوسيع نطاق التطبيقات التي يمكن عمليا معالجتها. i* يمكن أن تستخدم لنموذج التبعيات المحلية بين الوكلاء وكيف ان عملية اختيار التصميم تؤثر على اهداف الوكلاء. طرق الوكيل الموجه أصبحت شائعة في هندسة البرمجيات، في الأطر المعمارية، وأطر النمذجة لمتطلبات الهندسة والتصميم. i* هي اللغة غير الرسمية المعتمدة على الرسم البياني لمتطلبات المرحلة الهندسية المبكرة التي تدعم نماذج من التبعيات االمحلية بين الوكلاء وكيف ان عملية اختيار التصميم تؤثر على أهداف الوكلاء الوظيفية وغير الوظيفية . ConGolog هي الشكلية القائمة على المنطق للتعبير عن العمليات التي تحتوي على اكثر من وكيل. والاطارين يكمل كل منهما الآخر جيدا، وفي هذا العمل، استخدمنا مزيج من الإطارين في المتطلبات الهندسية. وسعت لغة الرسم البياني i*SR مع شروح مواصفات العملية، التي تسمح لنموذج SR للنظام ليتم تحسينه ومن ثم تحويله إلى نموذج ConGolog. التحويل يجب ان يلبي مجموعة من قواعد التحويل، والتي تضمن العناصر المرتبطة في النموذجين وأن النموذجين متناسقين.