Journal of Intelligent & Fuzzy Systems 27 (2014) 625–640
DOI:10.3233/IFS-131059
IOS Press

Choreographing agent encounters in the Semantic Web using rules

Kalliopi Kravari*, Nick Bassiliades and Christos Papavasileiou
Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki, Greece

Abstract. In order for automated agent-based e-Commerce transactions to blossom, well-defined, analyzable and easily customizable interaction protocols or choreographies of involved parties need to be developed. Although, several domain-depended protocols have already been developed, efficient methodologies and technologies for facilitating the definition, deployment, reuse and maintenance of interaction protocols should be developed. This paper proposes a rule-based, reusable, analyzable and easily comprehensible by the user choreography definition methodology, called K-SWAN. Ôhe proposed choreography scheme separates the definition of the agent shared interaction protocol from the private agent interaction strategy and enables agents to choose the appropriate protocol for the transaction, from a library of re-usable interaction protocols, and automatically combine it with their personal strategy, from a private library, by using SW technologies for both. Complying with K-SWAN methodology will let agents participate seamlessly in different interaction processes and/or modify their behavior with a minimal programming effort. Finally, this paper presents the integration of the K-SWAN methodology into EMERALD, a multi-agent knowledge-based framework based on SW standards, which maximizes reusability and interoperability of behavior between agents.

Keywords: Semantic Web, intelligent agents, reactive rules, agent interaction choreographies, agent policies

1. Introduction

E-Commerce has become a specialized discipline and it is clear that it will continue to grow [13]. However, the parties involved face difficulties in interacting correctly and efficiently, since they have to collect information, interact and safely execute transactions. To this end, well-defined interaction protocols or choreographies of involved parties are needed. However, the modeling, deployment and interoperability of protocols are really challenging tasks [15]. Several domain-depended protocols have already been developed, but efficient methodologies and technologies for facilitating the definition, deployment, reuse and maintenance of interaction protocols (or otherwise "choreographing the involved parties") should further be developed. Despite state of the art achievements, there is still a lack of a comprehensive integrated end-to-end solution. An open issue to this end is the use of Intelligent Agents (IAs), since their use could lead to an alternative intelligent interaction environment for the future [18].

IAs can benefit from Semantic Web (SW) technologies, such as RDF and RuleML for data and policy exchanges, facilitating interoperable interactions [14]. SW offers capabilities that can make services more interoperable, adaptable and cheaper to maintain. So far, sophisticated tasks, such as negotiation and brokering services, are already carried out efficiently by IAs

*Corresponding author. Kalliopi Kravari, Department of Informatics, Aristotle University of Thessaloniki, GR-54124 Thessaloniki, Greece. Tel.: +30 2310998231; Fax: +30 2310998433; E-mails: kkravari@csd.auth.gr, nbassili@csd.auth.gr (Nick Bassiliades), cpapavas@csd.auth.gr (Christos Papavasileiou).

since they are able to comprehend relevant information and satisfy task requests unsupervised.

In this context, this study proposes first of all the use of IAs, which use is constantly increasing as they are gradually enriched with SW technologies. In such an agent-based world, each agent encounter is characterized by the interaction protocol (or choreography as it is usually called in the web services world [15]), which is shared among participating agents, and its private "negotiating" (or otherwise) strategy. In other words, each agent is able to manage a private policy, a set of rules representing requirements, obligations and restrictions, personal data that meet its user's interests and a set of interaction rules that outline among others restrictions on the parties involved.

In order agents to reach their maximum efficiency, a well–formed modeling or choreography of the interactions between participants is needed. So far, having both protocol and strategy jointly hard-coded in each agent was a common practice; however, it was neither convenient nor flexible. Hence, the choreography of interactions should be separated from agents' strategies. This is a better practice since agent policy is private and any disclosure of it could lead to incalculable loss, thus it should stay hidden. On the other hand, each interaction protocol should be a common resource since agents must comply with the same protocol in order to interact. To this end, this study proposes a rule-based, reusable, analyzable and easily comprehensible by the user choreography definition methodology, called K-SWAN that promotes interaction automatization and agent usage.

The proposed choreography scheme separates the definition of the agent shared interaction protocol from the private agent interaction strategy and enables agents to choose the appropriate protocol for the transaction (from a library of re-usable interaction protocols) and automatically combine it with their personal strategy (from a private library) by using SW technologies for both. Hence, this article also proposes the use of SW languages for representing both protocol and strategy in addition to separating them. Complying with K-SWAN methodology will let agents participate seamlessly in different interaction processes and/or modify their behavior with a minimal programming effort.

Although the proposed methodology is a fully automated procedure for choreographing SW agent transactions, an appropriate framework providing compliance with the proposed SW technologies should be used. Hence, this article presents the integration of the K-SWAN methodology into EMERALD, a multi-agent knowledge-based framework [10] based on SW standards, which maximizes reusability and interoperability of behavior between agents.

The rest of the paper is structured as follows: Section 2 gives an overview of the approach, while Section 3 overviews EMERALD and presents the integration of the K-SWAN methodology in the system. Section 4 illustrates two use case scenarios based on the FIPA CNet and Brokering Protocol, demonstrating the potential of the approach. The paper concludes with references to related work, conclusions and directions for future improvements.

2. The K-SWAN methodology

An agent interaction involves private strategies that describe what parties want to accomplish in the particular encounter and public protocols that involve two or more agent roles and address specific purposes, focusing on the rules of encounter and omitting individual agent decision making details. Our methodology, called K-SWAN, is based on the claimed assumption that both strategies and protocols can be described using rules. Hence, they have to be separated and expressed in an appropriate (possibly different) rule language that will enable a knowledge-based agent interaction management approach.

2.1. Reaction RuleML

Studying web interactions reveals two important factors; efficient message exchange and flexible representation of the interaction rules. Hence, this study proposes the use of the Reaction RuleML language for expressing both the protocol and the strategy rules [5]. Reaction RuleML is a general, practical and user-friendly language and rule interchange format. It is based on RuleML, a unifying family of XML-serialized rule languages that covers the entire rule spectrum, from derivation to reaction rules. It is used for research and industry purposes while it has already built interoperation bridges between other rule languages. Specifically, it covers constructs for complex events, actions and states/fluents/transition definition and processing/reasoning for different derivation rule, production rule and reaction rule programs.

This rule language was chosen because it is flexible in rule representation and its syntax supports a message structure that can include all message modules provided by the FIPA specifications provided by FIPA,

```
cfp
:sender (agent-identifier: name "..")
:receiver (set(agent-identifier :name "..))
:content ".."
:ontology ontology1
:language fipa-acl
:protocol fipa-".."-protocol
:conversation-id conv0001
```

Fig. 1. Message structure according to FIPA.

```
<Message mode="outbound" directive=".">
   <oid><!--conversation ID--></oid>
   <protocol><!--protocol--></protocol>
   <sender><!--agent/service--></sender>
   <receiver><!--agent/service--></receiver>
   <content><!—message payload--></content>
</Message>
```

Fig. 2. Message structure in Reaction-RuleML syntax.

a widely known IEEE Computer Society organization that promotes agent-based technology [19]. Figures 1–2 present the similarity of message structure in RuleML and FIPA; both contain predicates for *Sender*, *Receiver*, *Content (Payload)*, *Protocol* and *Conversation Identifier* (*oid* and *conversation-id*).

Using appropriately these structures, agents are able to exchange from simple facts to rulebases (sets of rules), while due to the *conversation-id* module, agents will be able to get involved in longwinded and usually asynchronous communications, being flexible. In this context, the Reaction RuleML *SendMsg* and *rcvMsg* predicates are considered as appropriate for message exchange in any agent interaction:

```
sendMsg(XID, Protocol, Agent, Performative,
       Payload|Context)
rcvMsg(XID, Protocol, From, Performative,
       Payload|Context)
```

where *XID* is the *conversation-id*, *Protocol* is the specific type of the communication protocol, *Performative* is the type of the message, while *Payload* is the content of the message. Additionally, Reaction RuleML enables two types of rules, namely production and reactive. The former are used for agents' strategy, allowing them to act according to their user's will while reactive rules are used for the protocol, allowing agents to adjust to their partner's behavior, based on events related to message exchanges.

2.2. Protocol and strategy libraries

Having protocols and strategies separated and expressed in a language like Reaction RuleML enables flexibility and reusability. In accordance to the general principle of information reuse, reference libraries of rulebases could be an appropriate solution for having the available data easily accessible. A public library containing rulebases that represent agent interaction protocols and a private library (for each agent) containing similar rulebases for agent interaction policies could implement the idea above. The former will ensure that protocols are well-defined and can be easily accessed by any agent or service whereas the latter will allow each agent optimally and privately to manage its strategies. Following this philosophy, we propose the use of both a public protocol library and a private strategy library where protocols and strategies expressed in Reaction RuleML will be stored.

In other words, separating each agent's private strategy from the protocol leads to two main rule sets. The first is related to the Strategy (agent's personal policy) while the second is related to the Protocol. The Strategy rule set defines the agent's personal preferences whereas the Protocol rule set mainly choreographs message exchange among agents. Hence, whenever a new agent interaction incident occurs each agent should use the protocol library to choose the most appropriate protocol (rule set) for the transaction, ensuring that it will act properly throughout the overall procedure. Additionally, agents should use their private library to choose an appropriate strategy (rule set) that could maximize their utility.

Specifically, we propose the use of a public protocol library that will retain each available protocol with its name and information about how many times and when it was used. Hence, agents will identify among others popular protocols or newly available, acting adaptively upon each case. For instance, an old and widely used protocol should be more reliable and well-formed than a new one. Hence, agents will be able to adjust their strategy accordingly, becoming for instance more cautious. On the other hand, an old and not used lately protocol should be outdated. This knowledge can save them a lot of trouble, time and money. Additionally, we propose the use of a private strategy library where each agent will name and store its strategies along with information about its use (when, in which case and for which protocol) and its personal opinion about it, here called rank.

Strategies should also be reused but having stored an old strategy does not mean that it is successful. Agents should evaluate their strategies for future use, hence rank is needed. Usually agents base their evaluation on qualitative values like unacceptable or poor. In this context, we firstly associate each qualitative value

Table 1
Qualitative to quantitative evaluation values

Evaluation values		Qualitative	Quantitative
Qualitative	Quantitative	insignificant	0
poor	−1	good	1
unacceptable	−2	excellent	2

```
(deftemplate ACLMsg
   (slot communicative-act) (slot sender)
   (multislot receiver) (slot reply-with)
   (slot in-reply-to) (slot envelope)
   (slot conversation-id) (slot protocol)
   (slot language) (slot ontology)
   (slot content) (slot encoding)
   (multislot reply-to) (slot reply-by))
```

Fig. 3. Message structure in JESS syntax.

with a numerical value (Table 1) which ranges from −2 (absolutely negative) to +2 (absolutely positive), with 0 indicating insignificance. We use both positive and negative values in order to reflect the win or loss feeling after an encounter as happens in real life. Hence, we define as rank the score of each used strategy while a newly stored strategy will get the default 0 value. Each time a strategy is used, it is evaluated and the value is added to the stored rank.

A strategy's rank may be reduced since the sum value could be negative. Hence, using this information any private agent algorithm could deal with the strategy selection issue. For instance, some could always choose the strategies with the highest rank while others could prefer the newest strategy that exceeds a rank threshold. A variety of such algorithms, which are out of the scope of this study, could be used by an agent in order to build its behavior over time depending on its user's personal perception. However, this does not mean that the encounter will always be satisfied but it will have the best possible strategy among the available strategies.

2.3. Combining protocol with strategy

Agents' final behavior is a combination of protocol and strategy. Hence, the two rule sets have to be combined and executed in a compact way that will let agents react to their environment knowledge using their behavior patterns. To this end, the FIPA compliant JESS execution engine and language was chosen [7]. JESS has become popular in agent programming community since it was integrated by JADE [2], a reliable and widely used multi-agent framework. It is considered as a very expressive language that can express complex logical relationships with very little code. Figure 3 presents part of message structure in JESS syntax, indicating its FIPA compliance.

In order to have a single rule set in JESS syntax from the separated Reaction RuleML sets, domain depended XSLT transformations are used. Following our methodology, we set a public XSLT library where all the available XSLT transformations will be stored. A new library was needed since the available protocol library cannot be used for both protocols and XSLT transformations. Although, each transformation is associated with a specific protocol, each protocol can be used in many cases hence a number of XSLT stylesheets will be available for that protocol. Hence, each agent, as soon as it chooses the necessary protocol and its preferable strategy, will get an appropriate XSLT stylesheet from that library and use it in the protocol-strategy fusion process (Fig. 4). Notice that each protocol in the library should have at least an associated transformation.

3. Knowledge-based implementation

For practical grounds, this methodology was integrated into EMERALD [10], maximizing its efficiency. In this section we briefly review the features of EMERALD needed for comprehension.

3.1. EMERALD

EMERALD is a multi-agent knowledge-based framework built on JADE and based on SW and FIPA standards that enables reusability and interoperability of behavior between agents. EMERALD supported so far the implementation of various applications, like negotiation and brokering [10–12]. It provides a generic, reusable agent prototype (Fig. 5) for knowledge-customizable agents (KC-Agents), consisted of agent models (KC Models), a directory service (Advanced Yellow Pages Service) and external methods (Basic Java Library). AYPS provides a service that groups and sorts the registered services according among others their domain and type, allowing agents to make complex queries and receive the available services. Concerning KC Model, AYPS returns the providers as Jess facts with a designated format: *(service_type (provider provider_name))*.

Agents that comply with this prototype are equipped with a Jess rule engine [7] and a knowledge base (KB) that contains environment knowledge (in the form of facts), behavior patterns and strategies (in the form rules). Hence, the abstract specification of the prototype

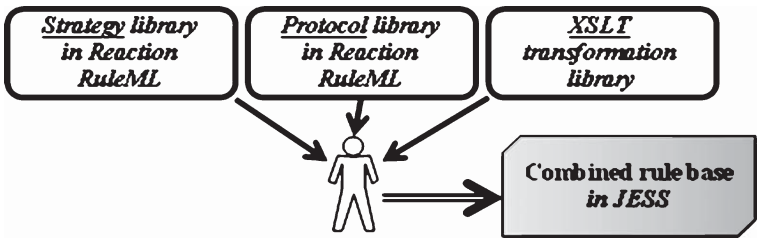

Fig. 4. Combing protocol with strategy.

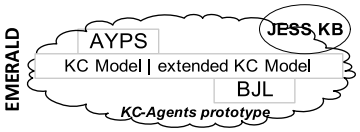

Fig. 5. KC-Agents prototype abstract architecture.

contains facts and rules; the generic (reaction) rule format is *result* $\leftarrow$ *rule* (*preconditions*). The agent's internal knowledge is a set of facts: $F = F_u \cup F_e$, where $F_u = \{f_{ul}, \ldots, f_{uk}\}$ are user-defined facts and $F_e = \{f_{el}, \ldots, f_{em}\}$ are environment-asserted facts. Agent behavior is a set of potential actions-production rules $P = A \cup C$, where A are fact derivation rules and C are communication rules, while $ACLMsg$ is a Jess template for ACL messages:

$$A = \{a \,|\, f_e \leftarrow a(f_{u1}, \ldots, f_{un}) \wedge \{f_{u1}, \ldots, f_{un}\} \subseteq F_u \wedge f \in F_e\}$$

$$C = \{c \,|\, ACLMsg \leftarrow c(f_1, \ldots f_p) \wedge \{f_1, \ldots, f_p\} \subseteq F\}$$

Using this prototype offers advantages, such as reusability, and interoperability of behavior between agents, as opposed to having behavior hard-wired into the agent's code.

Additionally, EMERALD adopts a variety of reputation mechanisms (e.g. [8]) that can also be used in strategy/protocol evaluation and handles reasoning interoperability. Since agents do not necessarily share a common rule or logic formalism, it is vital for them to find a way to exchange their position arguments seamlessly. To this end, EMERALD proposes the use of Reasoners [11], which are actually agents that offer reasoning services to the rest of the agent community. This approach does not rely on translation between rule formalisms but on exchanging the results of the reasoning process of the rule base over the input data. The receiving agent uses an external reasoning service to grasp the semantics of the rule base, namely the set of entailments of the knowledge base. Currently, EMERALD implements a number of Reasoners that offer reasoning services in two major reasoning paradigms: deductive and defeasible logic. Following these specifications EMERALD commits to SW and FIPA standards. It uses among others the RuleML language [5] since it has become a de facto standard and the RDF model [17] for data representation; both for the agents' internal knowledge data and the reasoning results generated during the process, as used in contract agreement interactions in [12].

3.2. Integrating K-SWAN methodology to EMERALD

EMERALD's KC-Agents prototype was extended to implement the K-SWAN agent choreography methodology. The set of derivation rules A is now:

$$A = \{a \,|\, u_1; u_2; \ldots; u_m \leftarrow a(f_1, \ldots, f_n) \wedge \{f_1, \ldots, f_n\} \subseteq F \wedge \{u_1, \ldots, u_m\} \subseteq F\}$$

Rules can now derive (more than one) new (or update existing) user-defined facts which can further trigger other rules, and so on so forth. In this way more complex decision making algorithms for the agent strategies can be implemented. Furthermore, the set of facts now also includes facts about messages F_m received from other agents: $F_m = \{f_{m1}, \ldots, f_{mk}\}$, $F = F_u \cup F_e \cup F_m$. User-defined facts can be: a) interaction management related parameters F_{IM}, such as conversation id, number of participants, etc., b) interaction content related parameters F_{CM}, such as the domain of a negotiation or other interaction (e.g. selling laptops online), the content of exchanged messages (proposed offer, accepted deals), c) current state of the interaction protocol F_{ST}, d) information exchanged between the strategy and protocol rulesets, through so called API facts F_{API}, and e) other general-purpose facts

$F_G : F_u = F_{IM} \cup F_{CM} \cup F_{ST} \cup F_{API} \cup F_G$.

The exact number and types of facts F_{IM} and F_{CM} depend on the type of interaction (e.g. negotiation, brokering, contract net, etc.) and is specified in the template of the protocol library. Also the number of states that the protocol goes into depends on the type of the interaction. Usually there is one state for each message exchange (pair of messages) of the protocol for each interaction participant. Furthermore, there are states F_{sST} involving the strategy of the agent, which are triggered when the protocol module needs to switch to the strategy modules whenever decision making is needed in order to formulate an answer to a received message or to initiate a new message exchange. Thus: $F_{ST} = F_{pST} \cup F_{sST}$, where F_{pST} are protocol states. The API facts contain the type and the content of the messages that the agent strategy ruleset infers to be sent by the protocol ruleset, so it needs a way to communicate such a decision:

F_{API} = {send(type, content, receiver, conversation - id)}

The equivalent "interface" facts between the protocol and the strategy are the interaction content related facts F_{CM}.

As explained previously, the agent's ruleset is composed by merging the public protocol ruleset P_p with the selected private strategy ruleset P_s : $P = P_p \cup P_s$. The two merged rulesets are composed as follows: $P_p = A_p \cup C_p$ and $P_s = A_s$, i.e. strategy does not contain agent communication rules. The components of the two merged rulesets are further defined as follows:

$$\begin{aligned} A_p = \{a_p \,|\, & u_1; u_2; \ldots; u_m \leftarrow a_p(f_1, \ldots, f_n) \\ & \wedge \{f_1, \ldots, f_n\} \subseteq F_{pST} \cup F_m \cup F_{IM} \\ & \wedge \{u_1, \ldots, u_m\} \subseteq F_{ST} \cup F_{IM} \cup F_{CM}\} \\ C_p = \{c_p \,|\, & ACLMsg \leftarrow c_p(f_1, \ldots, f_p \\ & \wedge \{f_1, \ldots, f_p\} \subseteq F_{pST} \cup F_{IM} \cup F_{API}\} \\ A_s = \{a_s \,|\, & u_1; u_2; \ldots; u_m \leftarrow a_s(f_1, \ldots, f_n) \\ & \wedge \{f_1, \ldots, f_n\} \subseteq F_{sST} \cup F_{CM} \cup F_G \\ & \wedge \{u_1, \ldots, u_m\} \subseteq F_{ST} \cup F_{API} \cup F_G\} \end{aligned}$$

The above can be explained as follows: the protocol may initiate a communication c_p because it has reached some state (F_{pST}), taking also into account interaction management parameters (F_{IM}) and/or triggered by the strategy ruleset (F_{API}). The protocol also waits for messages (F_m), expected at some state (F_{pST}), and based on the interaction management parameters (F_{IM}), it may advance to the next state (protocol or strategy, F_{ST}) or it may update the interaction management parameters (F_{IM}) or the interaction content parameters (F_{CM}). On the other hand, the strategy based on the current state (F_{sST}), it checks the parameters of the content of the interaction (F_{CM}) or an intermediate result/initial data of the decision making process (F_G) and decides either to advance to the next state (protocol or strategy, F_{ST}) or to formulate a message that the protocol should send (F_{API}) or just to store an intermediate derived fact that could be used later by another rule of the strategy ruleset (F_G).

So far KC-Agents were limited in receiving one file containing both strategy and protocol. The extended (K-SWAN) KC-agent receives two separate files one for the protocol and one for the strategy or just a set of keywords letting agent act independently. Hence, whenever protocol or strategy is modified, no extra programming cost is needed. The agent will retrieve the appropriate (new) files from the corresponding libraries. The transformation and merging of them will be executed automatically. Following this approach new behaviors and protocols can be added at any time to the libraries for future use.

Hence, three libraries were set as already described; the private strategy, the public XSLT and the public protocol library. The strategy library belongs to a specific agent and retains its strategies. Each record has the following properties; a name for the strategy, date that it was added to the library, the strategy itself (rulebase), description, date and case of use (for each time it was used) and its rank value. Currently, the rank variable holds just the final updated value, however all rank values in respect to time could be stored in future in order to check the fluctuation of the value over time. Valuable conclusions can be derived from studying it; defining if the decision making process of a strategy is getting outdated or not.

On the other hand, the public protocol library is managed by an independent agent and it is accessible by all interacting agents. It holds each available protocol with the following properties; an official name for the it, date that it was added to the library, the protocol itself (rulebase), date of use, case of use, agents involved and finally times of use (numerical value). Additionally, the independent agent manages a public XSLT library that holds each available transformation with the following properties; a name, date that it was added to the library, the transformation itself, date of use, case of use and times of use.

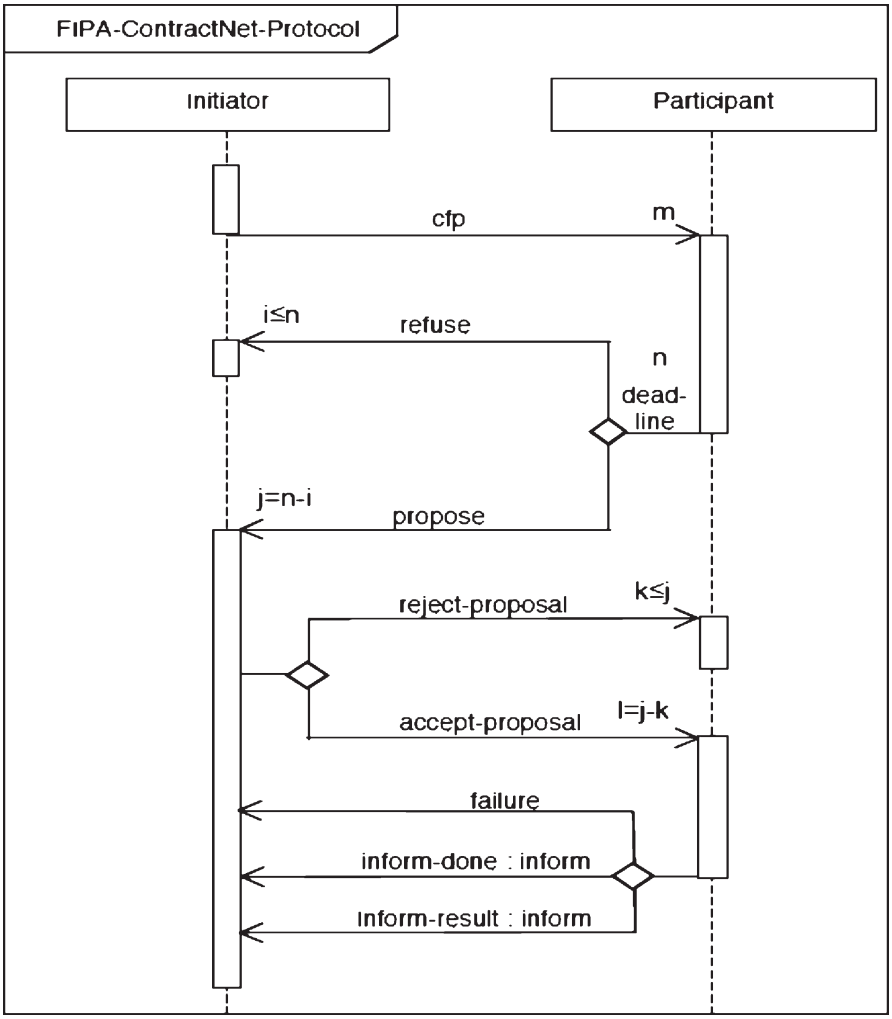

Fig. 6. Fipa Contract Net Interaction Protocol.

Each time, the user provides a file containing keyword(s), describing the case that it is interested in; an internal algorithm automatically looks up that keyword in a case list and retrieves an appropriate protocol. A protocol can be used in many cases. Yet, each case is associated with just a protocol. Similar, each strategy is associated with just a case (and thus a protocol). In this context, the prototype provides an expandable list associating cases (e.g. house brokering) with protocols (e.g. FIPA Brokering Interaction Protocol) and hence strategies. However, since a protocol describes more than one role, each agent has to choose the appropriate role for the case. The involved agents then agree on the protocol that will be used throughout their encounter. Yet, if an agent has to interact with a party, e.g. an online shop that enforces a predefined protocol the above procedure is omitted.

Additionally, having the protocol and the case used, each agent searches its personal library to find a strategy previously used in a similar case with a satisfying rank. Many criteria can be used in this selection algorithm; for instance the newest strategy associated to the desired case with a high rank. To this end, the extended prototype provides an internal algorithm that associates the agent's qualitative opinion to a quantitative value, as described in Table 1. Then, that value is used to update and store the new rank value in the library. Finally, a similar search is carried out in the XSLT stylesheet library in order to get the appropriate transformation. Then, the involved

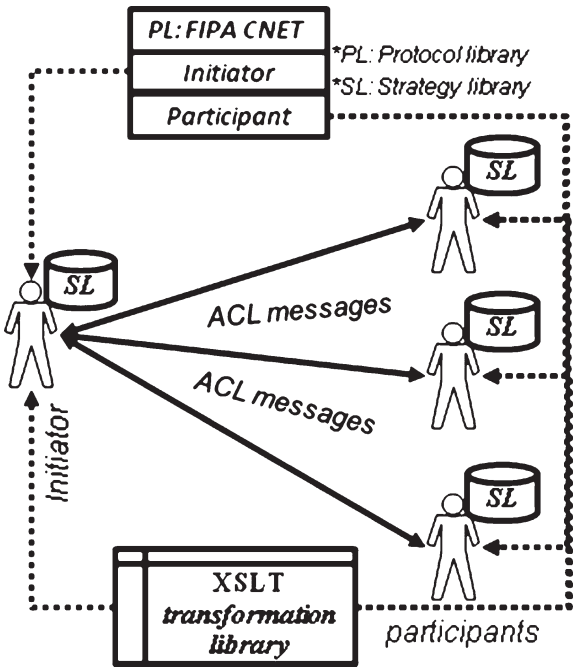

Fig. 7. Fipa CNET use case abstract overview.

parties have what they need to interact properly and proceed.

4. Use cases

Use case scenarios based on the FIPA Contract Net and Brokering Protocol were implemented using the K-SWAN methodology and are presented next in order to demonstrate the potential of our proposal.

4.1. FIPA contract net interaction protocol

The Contract Net Protocol (CNET) is probably the most widely used protocol, firstly introduced by Smith [16]. In CNET negotiation is considered as a two-way communication in which an agent evaluates the offer of assigning a contract or receiving one from its own perspective depending on its role. Although CNET was proved valuable in a variety of situations, it had to be modified in order to reflect changes in agent technology. In this direction, FIPA provides the FIPA Contract Net Interaction Protocol (Fig. 6) [19]. Based on this protocol, a scenario is presented clarifying why protocols and strategies should be separated and how an automated combination procedure saves time and programming effort.

This standardized protocol has been used over time as the basis for a variety of use cases. According to the FIPA specification in the contract net interaction protocol, one agent (Initiator) takes the role of the manager who wishes to have some task performed by one or more other agents (Participants) and further wishes to optimize a function that characterizes the task. This characteristic could be the minimum price or the soonest completion time. For a given task, the Initiator has to send a call for proposal message communicating its request. Next, any number of the Participants may respond positively; the rest must refuse. Negotiations then continue with the Participants that accepted the call. A positive response however is not a strict acceptance but rather a counter proposal. Hence, the Initiator has to evaluate the offers and ignore the refusals. Finally, it has to accept the best offer by sending back an acceptance message whereas reject messages should be send to the rest.

4.1.1. Use case implementation

A scenario based on this protocol was implemented in EMERALD, using the K-SWAN methodology, involving four parties; an initiator and three other participants (Fig. 7). All the involved parties comply with the new prototype provided by EMERALD, hence protocol and strategy are automatically combined. Specifically, the Initiator is interested in the best offer for a laptop; hence an appropriate strategy in RuleML, based on the keywords provided by its user and the highest rank value, is retrieved from its library. Next, it finds the FIPA CNet Protocol in RuleML for the initiator role in the protocol library. It also retrieves the XSLT stylesheet from the XSLT library. Then, the procedure for combining the protocol with the strategy is executed and Initiator is ready to start acting upon its goals.

The Initiator uses the AYPS service provided by EMERALD in order to locate potential e-shops. Eventually, it gets three represented by the participant agents. Following the protocol, the Initiator sends a CFP (Call-for-proposal) message containing the name of the used protocol, the name of the product and a desired price in order to initiate the negotiation procedure. Next, it waits for their response; either positive (PROPOSE) or negative (REFUSE). The participant agents have to decide if they are interested in interacting with the Initiator agent. In this use case they all are, because they are e-shops and their main strategy is to interact with any potential buyer. Hence, they retrieve the appropriate protocol, named in the CFP, from the public library for the participant role and the corresponding XSLT stylesheet

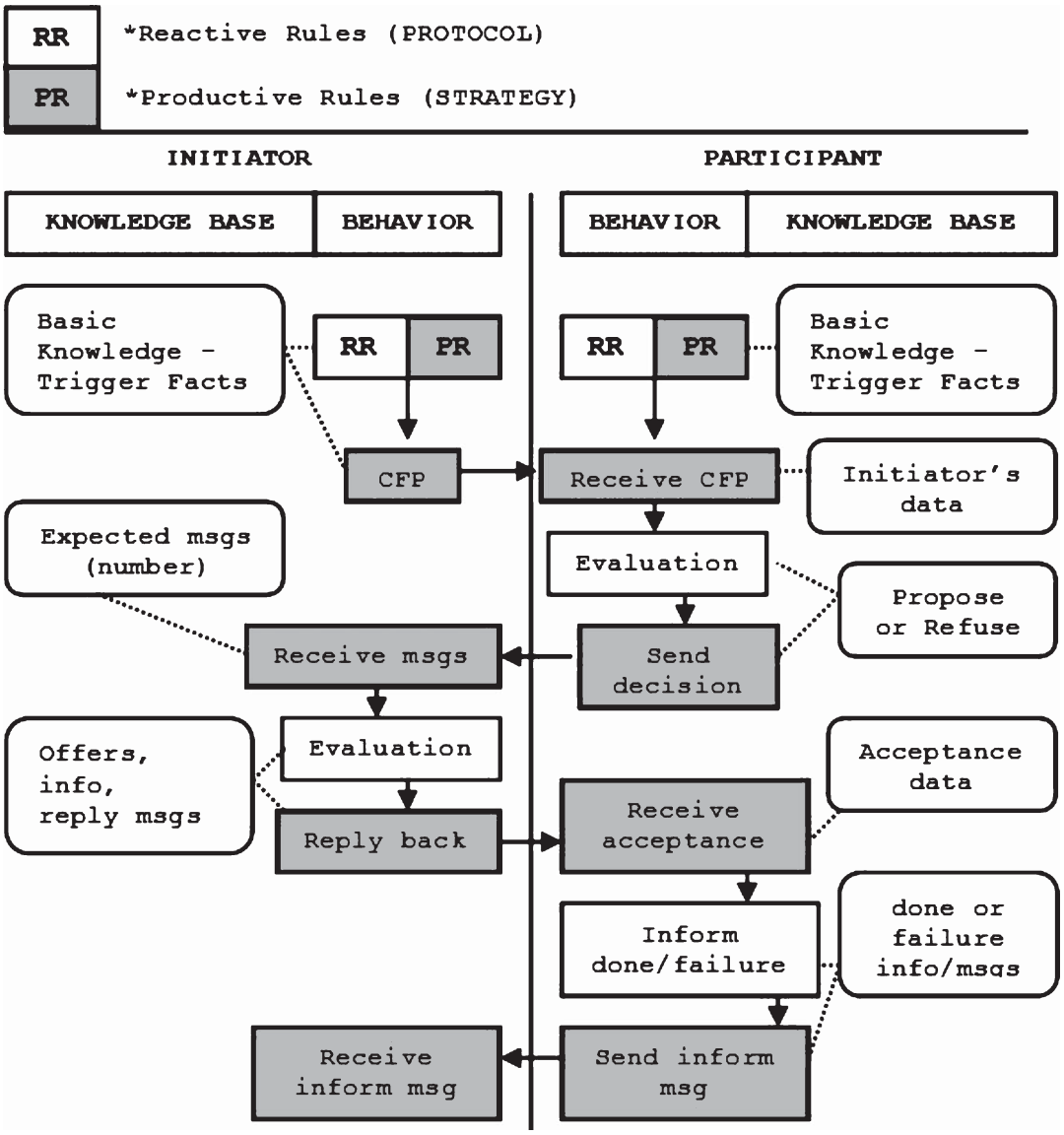

Fig. 8. Flow of rule activation, triggering and execution.

from the library. Next, they search their strategy library in order to find an appropriate for the case strategy with the highest rank value. Using the KC-Agents automated procedure, strategies are combined with the protocol.

Now being ready, they have to confirm that the product is available and act accordingly; refusing the call or proposing an offer. Since a positive response is not a strict acceptance but rather a price proposal, the participants may reply with proposal rather than just accept the call. The Initiator by its side has to evaluate the offers and ignore the refusals. According to the protocol, next it has to accept one offer by sending back an ACCEPT message whereas REJECT messages should be sent to the rest. The protocol defines the proper message exchange but it is the private strategies that define the decision making.

The Initiator's strategy, for instance, determines the agent's main restriction; the price offered by a participant should be lower than the price it is willing to pay. That price for the Initiator is the firstly indicated price in the CFP (here 300 Euros) plus an amount (here 50 Euros). If none of the participants fulfill this restriction, the Initiator will reject all of them. On the other

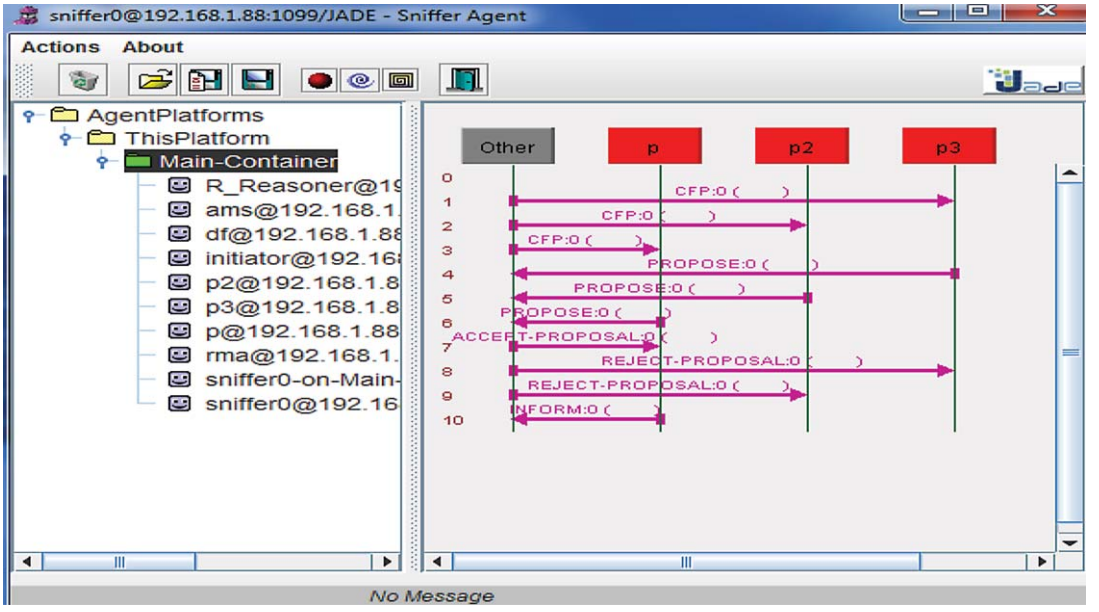

Fig. 9. Use case scenario message exchange in EMERALD.

hand, each participant has a list (set of facts called *Products*) that contains their available products accompanied with the product type (e.g. laptop) and its price (e.g. 500 Euros). Such a product in JESS is: *(products (type laptop) (price 200))*.

Using EMERALD and its extended prototype, we activated the four agents. Next, following the CNET protocol, a straight-forward procedure is performed from the first call to the final acceptance. Figure 8 presents an overview of the flow of rule activation and execution. A more detailed flow chart can be found in [9]. Here, the transaction was successfully completed when the Initiator chose the participant called *p* by sending an *Accept Proposal* message, as presented in the EMERALD's execution diagram (Fig. 9). Participant *p* sent back an *inform* message, which according to FIPA specifications is needed in order to verify that the final decision is received. Next, as soon as the interaction is finished, each agent evaluates the strategy and updates its rank value. Investigating evaluation algorithms and complex rank calculations is out of the scope of this study.

Finally, Reaction RuleML enables two types of rules; production for agents' strategy and reactive for the protocol. A reactive rule of a participant that when it receives a CFP message, it posts it as a fact in the agent's internal KB is presented in JESS (Fig. 10) and RuleML syntax (Fig. 11) for comparison purposes; the same rule is more compact in JESS than RuleML syntax.

```
(defrule receive-cfp
 (ACLMsg (communicative-act CFP)
  (conversation-id ?cid) (sender ?init)
  (protocol fipa-contract-net) (content ?c))
 ?x <- (state (state start))
=>
 (assert (callforp(cid ?cid)
  (content ?c) (sender ?init)))
 (modify ?x (id ?cid) (state check)))
```

Fig. 10. JESS Protocol Reactive Rule: Participant receives call.

Figure 12 presents a production rule of a participant that decides to participate in the bidding by sending a proposal, if the required product is available in its internal KB. Notice how the rules interact through a predefined set of fact templates that play the role of API (*callforp* and *send*), whereas *state* indicates the current protocol or strategy state.

The two rules are expressed as follows in the abstract KC-Agents/K-SWAN specification:

$$Callforp(cid, \ldots);\ state(check,\ cid) \leftarrow receive\text{-}cfp(state(start),\ ACLMsg(CFP, cid, \ldots)) \in A_p$$

$$send(PROPOSE,\ price,\ sender,\ cid);\ state(send_inf,\ cid) \leftarrow check\text{-}cfp1(state(check, cid),\ Callforp(cid,\ sender,\ product),\ product(product,\ price)) \in A_s$$

where $state(start) \in F_{pST}$, $state(check) \in F_{sST}$, $ACLMsg \in F_m$, $Callforp \in F_{CM}$, $send \in F_{API}$ and $product \in F_G$.

```
<Rule>
   <oid><Ind>receive_cfp</Ind></oid>
   <on>
      <Message mode="inbound" directive="CFP">
         <oid><Var>cid</Var></oid>
         <protocol>
            <Ind>fipa-contract-net</Ind>
         </protocol>
         <sender>
            <Var>initiator_name</Var> </sender>
         <content><Var>call</Var></content>
      </Message>
   </on>
   <if><And>
      <Atom><Rel>state</Rel>
         <slot> <Ind>state</Ind>
                <Ind>start</Ind> </slot></Atom>
      <Atom><oid><Var>start_state</Var></oid>
         <Rel>state</Rel>
         <slot> <Ind>state</Ind>
                <Ind>start</Ind> </slot></Atom>
   /And></if>
   <do><Succession><Assert>
      <Atom><Rel>callforp</Rel>
         <slot> <Ind>cfp</Ind>
                <Var>call</Var>  </slot>
         <slot>
            <Ind>cfp </Ind>
            <Var>initiator_name</Var> </slot>
         <slot> <Ind>cfp_cid</Ind>
                <Var>cid</Var></slot></Atom>
   </Assert>
   <Update>
      <Atom><oid><Var>start_state</Var></oid>
         <Rel>state</Rel>
         <slot> <Ind>state</Ind>
                <Ind>check</Ind> </slot>
         <slot> <Ind>id</Ind>
                <Var>cid</Var></slot></Atom>
   </Update></Succession></do>
</Rule>
```

Fig. 11. RuleML Protocol Reactive Rule: Participant receives call.

```
(defrule check-cfp1
 ?p <- (state (id ?cid) (state check))
 (callforp (cid ?cid)
  (sender ?name) (content ?prod))
 (products (type ?prod) (price ?price))
 =>
 (assert (send (act PROPOSE) (content ?price)
  (receiver ?name) (cid ?cid)))
 (modify ?p (state send_int) (id ?cid)))
```

Fig. 12. Production Rule (Strategy): Participant evaluates call and prepares PROPOSE response.

4.2. FIPA brokering interaction protocol

The FIPA Brokering Interaction Protocol [19] is quite popular, designed to support brokerage interactions. Generally speaking, a broker is an agent that offers a set of communication facilitation services to other agents using some knowledge about the requirements and capabilities of those agents. A typical example of brokering is one in which an agent can request from a broker to find one or more agents who can answer a query. The broker then determines a set of appropriate agents to which to forward the query, sends the query to those agents and relays their answers back to the original requestor. The use of brokerage agents can significantly simplify the task of interaction with agents in a multi-agent system. Additionally, brokering agents also enable a system to be adaptable and robust in dynamic situations.

More specifically, the Initiator of the brokering interaction (Fig. 13) begins the interaction with a proxy message which contains: a reference to the target, the request and a set of proxy conditions such as the maximum number of agents to which the message should be forwarded. The Broker processes the request and makes a decision whether to agree or refuse and replies accordingly. Once the Broker has agreed to be a proxy, it locates agents according to the description from the proxy message. If no such agents can be found, the Broker returns a *failure-no-match* and the interaction terminates. Otherwise, the Broker may modify the list of matching agents based on the proxy conditions specified by the Initiator and begins interactions with the target agents (sub-protocol). When these interactions end (successfully or not), the Broker forwards the final response to the Initiator.

4.2.1. Use case implementation

A scenario based on the above protocol was implemented in EMERALD following the K-SWAN methodology, namely all parties comply with the new prototype. This scenario is loosely based on one of the most commonly used W3C's SWS usage scenarios [6]. We adopt the main idea of a travel agency that provides tourism services to end-users. Thus, we have a virtual travel agency, called VTA for short, which offers to clients the ability to book vacation services. These services can support booking of flights, hotels, rental cars, etc. Hence, this scenario involves: the travel agency, a customer and a number of target agents proving vacation services. The abstract overall process is presented in Fig. 14. The Initiator (here the customer) is interested in booking a vacation by targeting to the best combination of services and prices for its needs. Hence, an appropriate for the case strategy in RuleML with a rank value at least 1 is retrieved from its private library. Next, the Initiator searches the protocol library for the protocol posted by the Broker in order to comply with the same interaction protocol; the FIPA Brokering Interaction Protocol in RuleML for the initiator role. Additionally, it retrieves the XSLT stylesheet from the public library. Then, the procedure for combining the protocol with the strategy is automatically executed.

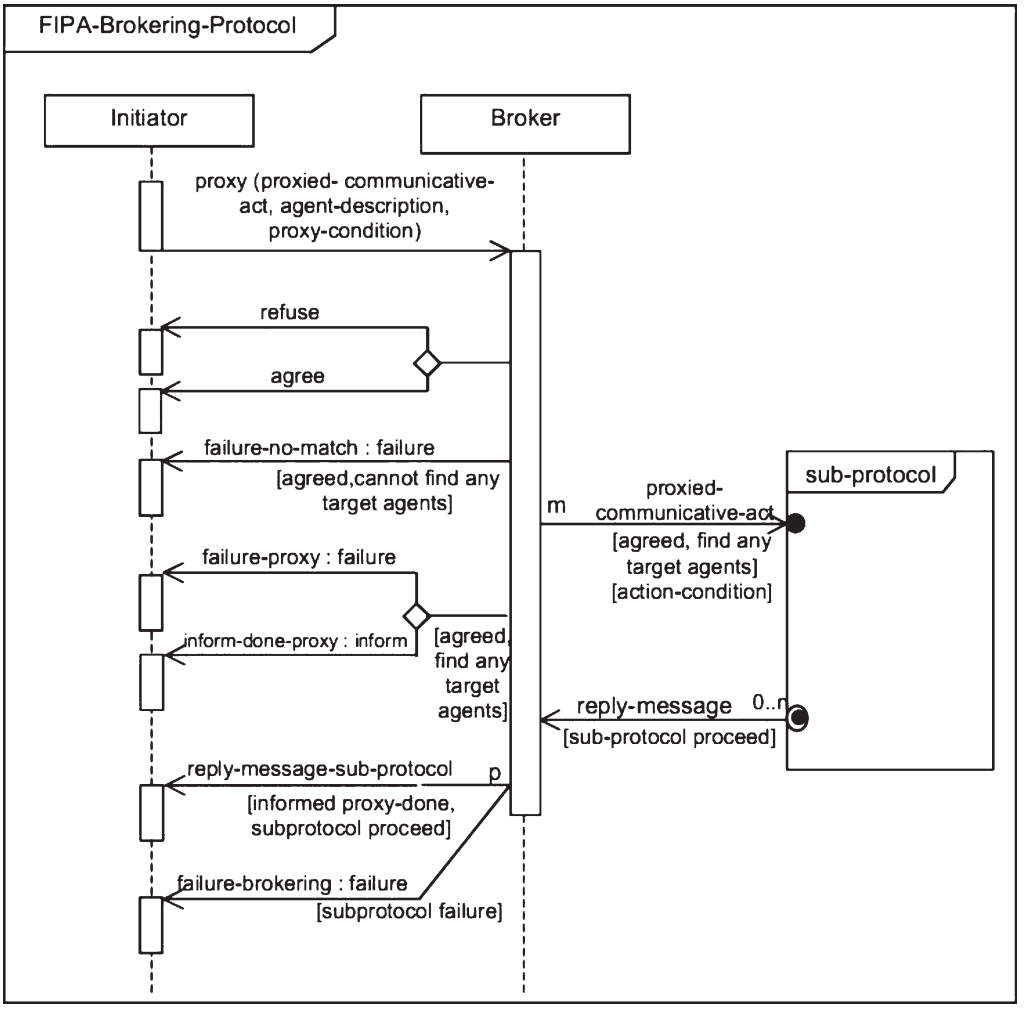

Fig. 13. FIPA Brokering Interaction Protocol.

The Initiator, following the protocol, sends a *proxy* message containing the desired services and some proxy conditions. Specifically, it is interested in travelling to Cuba during July and wants at least a 3* hotel, a two-way air ticket and a travel insurance. Concerning proxy conditions, it defines the minimum number of agents to which the message should be forwarded (here three) and an international travel insurance provider. Next, it waits for the response; either positive (AGREE) or negative (REFUSE). The Broker processes the request and decides to agree. It has already retrieved the Brokering protocol for the broker role and gets the corresponding XSLT stylesheet and a promising strategy (highly ranked - many times used) from its library. Strategy and protocol are combined by the automated procedure.

Being ready, the Broker informs the Initiator. At the same time, it locates agents per the description from the proxy message. Hence, the Broker, as soon as it accepts the job, locates five service agents that offer what it needs fulfilling Initiator's conditions and begins interacting with them. The service providers communicate with the Broker and prepare an offer. Having five different offers, the Broker forwards them to the Initiator that eventually chooses the cheapest. Figure 15 presents a flow overview of rule activation and execution. Interaction among Broker and the rest target agents is a recursive sub-protocol procedure. Since multiple agents match, Broker initiates multiple sub-protocols within the brokering protocol. Here, the Broker initiates five instances of the FIPA CNet protocol presented above. In this case, the Broker may

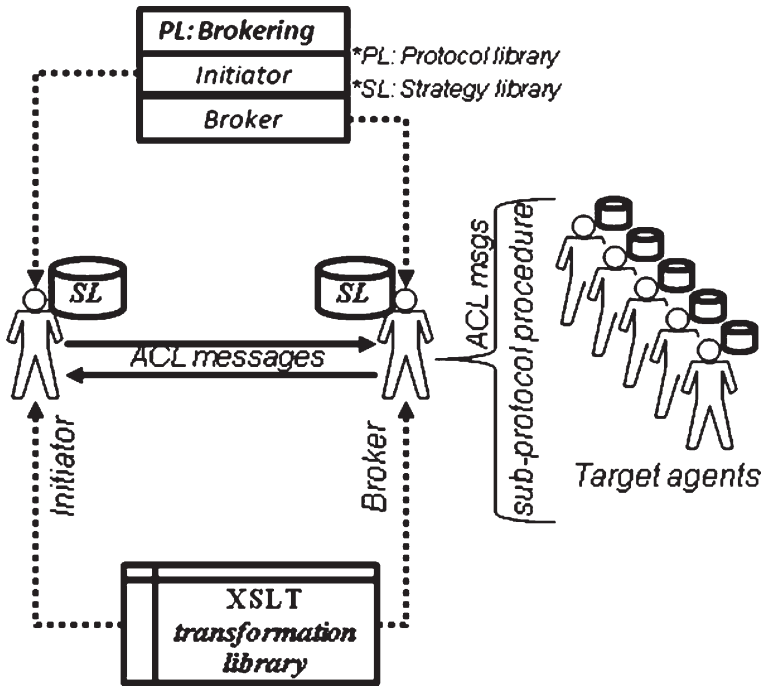

Fig. 14. Fipa CNET use case abstract overview.

collect the received responses and combine them into a single reply-message-sub-protocol, or may forward the reply-message-sub-protocol messages from the separate sub-protocols individually (this is the case here). Hence, describing in detail these cases is out of the scope of this example.

Finally, following the K-SWAN methodology, we have two types of rulesets in Reaction RuleML; production rules for the strategy and reactive rules for the protocol. Figure 16 presents a reactive protocol rule in JESS that is used by the broker to receive proxy requests and store them in order to forward them later if it decides so. The same rule is also presented in Fig. 16 in Reaction RuleML syntax. This rule allows the Broker to post as a fact in its internal KB every received proxy request and prepare for the next decision making step (agree or refuse).

This rule is expressed as follows in the abstract KC-Agents/K-SWAN specification:

$$receive\_proxy(cid, \ldots)); state(decide\_proxy, cid) \leftarrow$$
$$receive\text{-}proxy\text{-}request(state(start),$$
$$ACLMsg(proxy, cid, \ldots)) \in A_p$$

where $state(start) \in F_{pST}$, $state(decide\_proxy) \in F_{sST}$, $ACLMsg \in F_m$, and $receive\_proxy \in F_{CM}$.

5. Related work

To the best of our knowledge our proposed methodology is the first one to separate protocol and strategy using a knowledge-based approach. Nevertheless, [1] discusses rule-based price negotiation approaches in multi-agent e-commerce systems, supporting that rules are a feasible and scalable technology for automated negotiations. The authors summarize the state-of-the-art in rule-based approaches to automated negotiations and present some initial experimental results using a rule-based price negotiation mechanism. Yet, there is no implementation information about the mechanism, except that it is based on the JADE framework and JESS. Our work, on the contrary, concerns modeling a specific reusable SW-compliant procedure that could be used not only in price negotiation but also in any other protocol case. Additionally, K-SWAN methodology implementation is described in detail and it is available to use in EMERALD. Yet, both approaches deal with the idea of automating e-commerce transactions. They consider IAs and rules important for maximizing automation and efficiency while multi-agent systems are claimed to be one of promising technologies for achieving this goal.

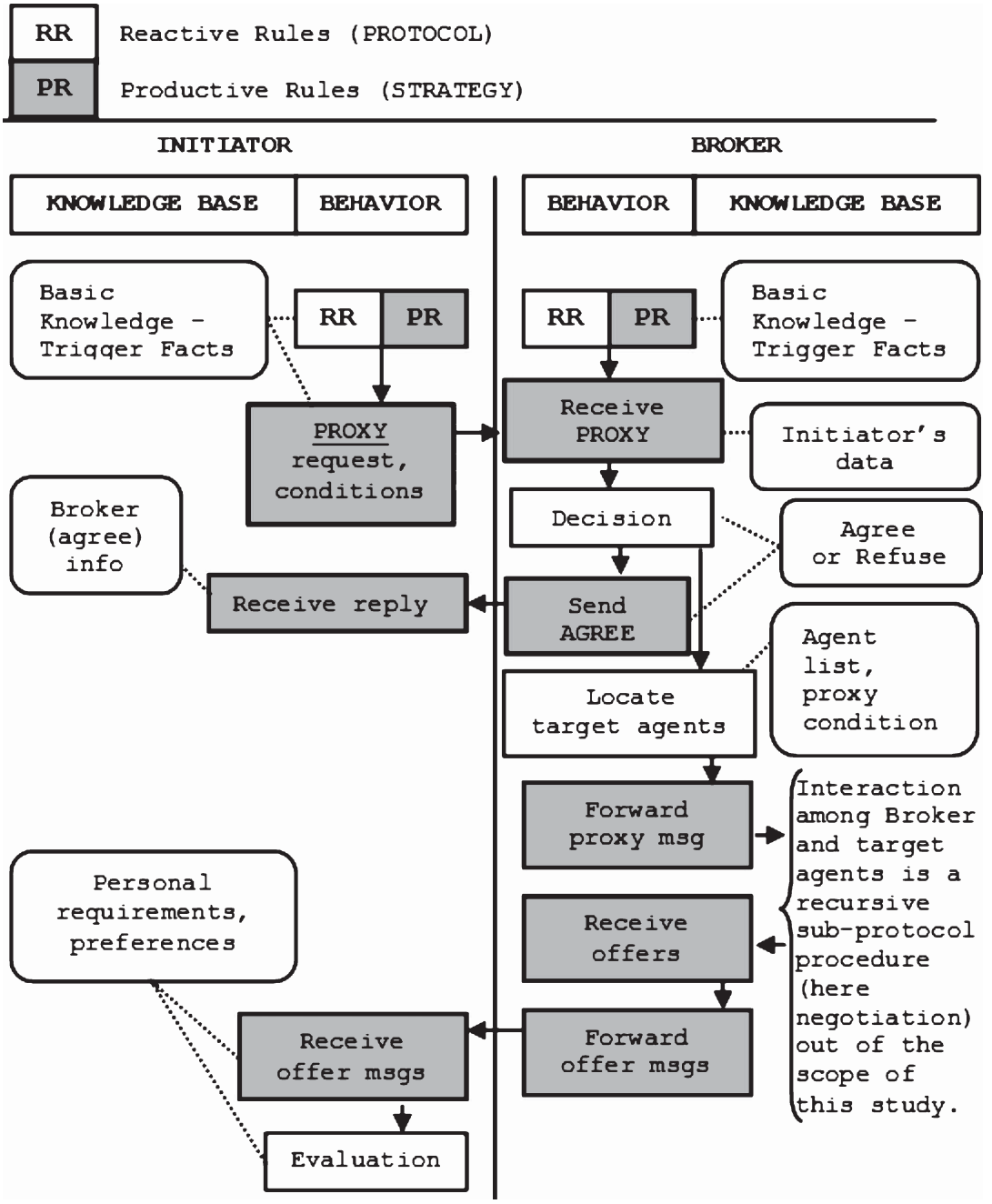

Fig. 15. Flow of rule activation, triggering and execution.

Concerning defeasible logic, a related approach is the DR-CONTRACT [3] architecture for representing and reasoning on e-Contracts in defeasible logic. It captures the notions relevant to execution and performance of e-Contracts in defeasible logics. The authors in order to integrate the DR-CONTRACT with SW technology use a RuleML extension that combines deontic notions with defeasibility and violations and RDF/XML syntax for its exported conclusions. Hence, although it focuses only on e-Contracts, omitting separation of protocols and strategies, it is a SW compliant approach like ours. We do agree in usefulness of rules and defeasible logic, although we do not use the deontic defeasible logic of violation. Moreover, our approach is not limited to the

```
(defrule receive-proxy-request
 (ACLMsg (communicative-act proxy)
  (conversation-id ?cid) (sender ?init)
  (protocol fipa-brokering) (content ?c))
 ?x <- (state (state start))
=>
 (assert (receive_proxy(cid ?cid)
  (content ?c) (sender ?init)))
 (modify ?x (id ?cid) (state decide_proxy)))
```

Fig. 16. JESS Reactive Rule: Broker receives proxy request.

use of defeasible logic or e-Contract transactions either. DR-CONTRACT is more like a defeasible reasoning engine but our K-SWAN methodology is an end-to-end procedure for any kind of agent transaction in the SW.

Concerning interoperability, Rule Responder [4] is quite similar to EMERALD. It builds a service-oriented methodology and a rule-based middleware for interchanging rules in virtual organizations, as well as negotiating about their meaning. It demonstrates the interoperation of distributed platform-specific rule execution environments, with Reaction RuleML as a platform-independent rule interchange format. Although, it supports only web services acting like agents, it deals with interoperability, reusability and even protocol-strategy separation issues like our approach. More specific, it has a similar view of rules, reasoning services and usage of RuleML but it is not FIPA compliant. Yet, Rule Responder is limited to representing virtual organizations while EMERALD enriched with the K-SWAN choreography methodology is able to support any agent community. However, they are both environments that support intelligent transactions in the sense that the parties involved, regardless they are represented by agents or services, can (re)use sets of rules and implemented procedures to behave and interact.

6. Conclusions and future work

The article argued that the massive growth of e-Commerce enabled new ways of transactions, which are vital but rather complicated. We addressed this problem by using IAs acting in the SW, as agents can perform the same tasks unsupervised. To this end, this article presented a modular and reusable framework using SW technologies, such as RuleML and RDF. Specifically, this article studied how separated interaction protocols and strategies can be combined through a rule-based choreography methodology, enabling reusability and thus agent participation in interaction processes without the need of reprogramming. Additionally, for

```
<Rule>
  <oid>
    <Ind>receive-proxy-request</Ind> </oid>
  <on>
  <Message mode="inbound" directive="proxy">
    <oid><Var>cid</Var></oid>
    <protocol>
      <Ind>fipa-brokering</Ind> </protocol>
    <sender>
      <Var>initiator_name</Var> </sender>
    <content><Var>call</Var></content>
  </Message>
  </on>
  <if>
    <Atom><oid><Var>start_state</Var></oid>
      <Rel>state</Rel>
      <slot>
        <Ind>state</Ind>
        <Ind>state</Ind> </slot></Atom>
  </if>
  <do><Succession><Assert>
    <Atom><Rel>receive_proxy</Rel>
      <slot>
        <Ind>content</Ind>
        <Var>call</Var>  </slot>
      <slot>
        <Ind>sender</Ind>
        <Var>initiator_name</Var> </slot>
      <slot> <Ind>cid</Ind>
             <Var>cid</Var></slot></Atom>
  </Assert>
  <Update>
    <Atom><oid><Var>start_state</Var></oid>
      <Rel>state</Rel>
      <slot> <Ind>state</Ind>
             <Ind>decide_proxy</Ind> </slot>
      <slot> <Ind>state_id</Ind>
             <Var>cid</Var></slot></Atom>
  </Update></Succession></do>
</Rule>
```

Fig. 17. RuleML Reactive Rule: Broker receives proxy request.

practical grounds this choreography methodology was integrated into EMERALD, a multi-agent knowledge-based framework, and two use case scenarios evaluated the approach.

As for future direction, our main interest is to provide a general-purpose framework for an automated end-to-end choreography management in multi-agent environments, letting agents maximize their autonomy, flexibility and efficiency. For this purpose, more protocols, like all FIPA compliant protocols, have to be integrated in the proposed framework. Additionally, we plan to use an ontology-based approach to ease the selection of the appropriate protocol according to the specific encounter properties, so that to extend our methodology functionality to non-expected cases.

Acknowledgments

This work is partially supported by the Greek R&D General Secretariat through a bilateral Greek-Romanian project.

References

[1] C. Badica, M. Ganzha and M.L. Paprzycki, Implementing rule-based automated price negotiation in an agent system, *Journal of Universal Computer Science* **13**(2) (2007), 244–266.

[2] F. Bellifemine, G. Caire, A. Poggi and G. Rimassa, JADE: A white Paper, *EXP in search of innovation* **3**(3) (2003), 6–19.

[3] G. Governatori and D.H. Pham, DR-CONTRACT: An architecture for e-contracts in defeasible logic, *Int J of Business Process Integration and Management* **4**(3) (2009), 187–199.

[4] H. Boley and A. Paschke, Rule responder agents framework and instantiations, *Semantic Agent Systems, Studies in Computational Intelligence* **344** (2011), 3–23.

[5] H. Boley, A. Paschke and O. Shafiq, RuleML 1.0: The overarching specification of web rules, *4th International Web Rule Symposium: Research Based and Industry Focused (RuleML'10)*, Springer, **6403** (2010), 162–178.

[6] H. He, H. Haas and D. Orchard, Web Services Architecture Usage Scenarios, W3C Working Group Note, available at http://www.w3.org/TR/ws-arch-scenarios/, 2004.

[7] JESS, the Rule Engine for the Java Platform, available at http://www.jessrules.com/, 2008.

[8] K. Kravari and N. Bassiliades, HARM: A hybrid rule-based agent reputation model based on temporal defeasible logic, 6th International Symposium, RuleML 2012, Springer, *LNCS* **7438** (2012), 193–207.

[9] K. Kravari, C. Papavasileiou and N. Bassiliades, Knowledge-based e-contract negotiation among agents using semantic web technologies, *Proceedings of the 5th International Conference on Computational Collective Intelligence Technologies and Applications (ICCCI 2013)*, 2013.

[10] K. Kravari, E. Kontopoulos and N. Bassiliades, EMERALD: A multi-agent system for knowledge-based reasoning interoperability in the semantic web, in: *Artificial Intelligence, Theories, Models and Applications, SETN 2010*, Springer, *LNCS* **6040/2010** (2010), 173–182.

[11] K. Kravari, E. Kontopoulos and N. Bassiliades, Trusted reasoning services for semantic web agents, *Informatica: Int Journal of Computing and Informatics* **34**(4) (2010), 429–440.

[12] K. Kravari, G.E. Kastori, N. Bassiliades and G. Governatori, Contract agreement policy-based workflow methodology for agents interacting in the semantic web, *Semantic Web Rules, Proc. 4th International Web Rule Symposium (RuleML 2010)*, Springer, *LNCS* **6403** (2010), 225–239.

[13] K. Laudon and C.G. Traver *E-Commerce 2013, 9/E, Prentice Hall*, New Jersey 2012.

[14] L. Feigenbaum, I. Herman, T. Hongsermeier, E. Neumann and S. Stephens, The semantic web in action, *Scientific American* **297** (2007), 90–97.

[15] M. Baldoni, C. Baroglio, A.K. Chopra, N. Desai, V. Patti and M.P. Singh, Choice, interoperability, and conformance in interaction protocols and service choreographies. *Proc. 8th Int. Conf. on Autonomous Agents and Multiagent Systems* **2**(2009), 843–850.

[16] R.G. Smith, The contract net protocol: High level communication and control in a distributed problem solver, *IEEE Transactions on Computer* **29**(12) (1980), 1104–1113.

[17] Resource Description Framework. (RDF), Model and Syntax Specification, available at http://www.w3.org/TR/PR-rdf-syntax/, 2004.

[18] S.J. Russell and P. Norvig, *Artificial Intelligence: A modern approach (2nd ed.)*, Prentice Hall, Upper Saddle River 2003.

[19] The Foundation for Intelligent Physical Agents (FIPA), FIPA Communicative Act Library Specification, available at http://www.fipa.org/specs/, 2003.