

CS 4983

Senior Technical Report

Evaluating Centralized, Hierarchical, and Networked
Architectures for Rule Systems

Benjamin Larry Craig
3155984

Supervisors:
Harold Boley
Michael Fleming

Cordinator:
Rick Wightman

Faculty of Computer Science
University of New Brunswick
Fredericton New Brunswick, Canada

November 13, 2008

Executive Summary

There are complex trade-offs between system designs that are either centralized or distributed in one of several ways. A centralized system has all of the functional software components on a single computer or server while a distributed system has components distributed over multiple computers or servers. Distributed systems require a communication topology for connecting the different components. A distributed system will be subdivided in the report into either a hierarchical or a networked architecture. A hierarchical distributed architecture employs a client-server-like topology with communication restricted to ‘top-down’ querying from the server to the clients and, conversely, ‘bottom-up’ query answering, as well as a single point of contact. A networked distributed architecture employs a peer-to-peer-like topology with query-answer communication allowed to flow between all peers, and does not have a central point of contact.

This report will present an evaluation of centralized, hierarchical, and networked architectures when used in rule-based systems with a focus on the communication topology. Our rule-based systems will be purely declarative, with communication restricted to query-answering (rather than, say, the execution of actions). Since a centralized system can be regarded as an extreme case of a distributed system, the three architectures can actually be considered as constituting one spectrum. OO jDREW, an engine operating on knowledge bases stored in RuleML, will be employed as an implemented use case for centralized systems. Rule Responder, connecting OO jDREW, Prova, and other engines, will be used to exemplify distributed systems, with the current implementation being hierarchical. Rule Responder’s development into a networked architecture will be explored to guide future implementations. The architectures will be compared with respect to evaluation criteria that were suggested by earlier work.

The evaluation of the three architectures based on our use cases will involve criteria such as knowledge maintenance issues in centralized vs. distributed systems, general performance issues, the fault tolerance of the distributed systems, the communication overhead coming into the distributed topologies, and the bottleneck issue in hierarchical systems. Our evaluation will employ empirical evidence obtained through practical benchmarking (centralized, hierarchical) and theoretical extrapolation (networked). The testing will be done on an application created with Rule Responder and OO jDREW that realizes query-answering in a symposium planner. This testing will explore suitable topologies for knowledge-based multi-agent systems.

Table of Contents

1	Introduction	4
2	Description of Technologies	5
2.1	RuleML Knowledge Representation in the Semantic Web	5
2.2	Distributed Systems	6
2.3	Hierarchical (Star Topology, Client-Server-Like Architecture)	6
2.4	Networked (Full/Partial Mesh Topology, Peer-to-Peer-Like Architecture) . .	7
2.5	Rule Engines: OO jDREW and Prova	8
2.6	Rule Responder	8
3	Knowledge Maintenance Issues in Centralized and Distributed Rule Sys-	
	tems	11
3.1	Organization of Distributed Knowledge	11
3.2	Module Boundaries	12
3.3	Centralized Knowledge Maintenance	13
3.4	Distributed Knowledge Maintenance	14
4	General Performance Issues of Distributed Topologies	15
4.1	Star Topology Advantages	16
4.2	Star Topology Disadvantages	16
4.3	Advantages of Peer-to-Peer Networks	17

4.4	Fault Tolerance of Distributed Topologies	17
5	Benchmarking Evaluation	18
5.1	Description of Use Case	18
5.2	Centralized Performance (OO jDREW)	19
5.3	Hierarchical Performance (Rule Responder)	20
5.4	Peer-to-Peer (Theoretical Efficiency Considerations)	22
6	Conclusion	24

List of Figures

2.1	Star Topology	7
2.2	Mesh Topology	7
2.3	Partial Mesh Topology	8
2.4	Rule Responder Architecture	9
5.1	Communication Sequence for RuleML-2008 use case	21

Chapter 1

Introduction

Distributed systems are becoming more important in computer science. The Semantic Web is also becoming an emerging technology in computer science. The Semantic Web improves the understanding of Web-based information. One way to give meaning to Web-based information is to store metadata (knowledge) about it as logic facts and rules. Such rule systems here are similar to pure Prolog specifications but are extended with declarative object-oriented features to be more expressive. Thus, our rule-based systems will be purely declarative, with communication restricted to query-answering (rather than, say, the execution of actions). The combination of a Semantic Web system and a distributed system into a single system thus creates a distributed rule-based system. Differences in maintainability, performance, etc. between a centralized system and various kinds of distributed systems are important when designing a Web-based rule system. Distributed systems require a communication topology because they operate over a set of computers, while a centralized system is run on a single computer. Within distributed systems our comparison will focus on centralized vs. distributed architectures.

Chapter 2

Description of Technologies

This chapter will discuss the different technologies presented in the report. Many different concepts and technologies are employed throughout the paper. This chapter will introduce each of the major concepts discussed.

2.1 RuleML Knowledge Representation in the Semantic Web

In a rule-based system the knowledge must be stored in a rule language. The knowledge used in the evaluation of OO jDREW and Rule Responder is stored in RuleML [Bol01]. RuleML is a flexible XML-based rule language that has the expressivity required by Rule Responder and OO jDREW. RuleML also has a presentation syntax known as POSL [Bol04]. POSL is a Prolog-like syntax that is human readable. The follow fact and rule is represented in POSL [BGT05]:

Fact:

```
spending(Peter Miller, min 5000 euro, last year).
```

Rule:

```
premium(?Customer) :- spending(?Customer, min 5000 euro, last year).
```

This fact in English means “Peter Miller spent at least 5000 euros last year”. The rule in English means “A customer is premium if they spent at least 5000 euros last year”. A rule engine like OO jDREW given this fact and rule would be able to derive that “Peter Miller” is a premium customer because “he spent at least 5000 euros last year”.

2.2 Distributed Systems

A distributed system is a set of computer processes that appear to the user as a single system [TvS]. The distributed system must look after the coordination of all of these processes and usually requires a middleware to do so [Arn]. An example middleware tool that will be discussed is the Mule enterprise service bus (Mule ESB) [BCC⁺]. Distributed systems are used to improve the performance and scalability of their centralized counter parts [Arn]. The components of a distributed system are connected with a topology. The topologies of interest will be discussed in next sections.

2.3 Hierarchical (Star Topology, Client-Server-Like Architecture)

Star topologies are the most commonly used topologies. The star topology (shown in figure 2.1) refers to a single-level hierarchy and not a multi-level hierarchy which is a tree topology. The star topology connects all nodes through a centralized hub. This is also commonly known as a client-server-like architecture. Where the server is the hub and all the clients are the outside spokes. A star topology is fault tolerant because if a spoke is broken then it will not affect the rest of the spokes [Tea]. If the centre hub is broken then the entire system is broken. Once a large amount of spokes are connected through the centralized hub, bottleneck issues can occur and when this occurs a different topology may perform better.

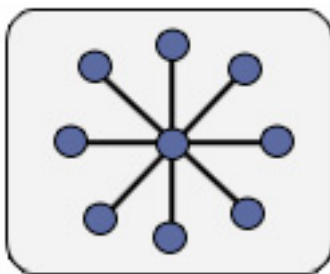


Figure 2.1: Star Topology

2.4 Networked (Full/Partial Mesh Topology, Peer-to-Peer-Like Architecture)

The full mesh topology (shown in figure 2.2) is a fully redundant topology, meaning that any node can send data to any other node and this ensures that there is always information flow to any node if a node fails [Tea]. The problem with a full mesh topology is that it requires many connections and is not practical when a large number of devices are on connected through the topology. The alternative to a full mesh topology is a partial connected mesh topology where only some nodes are connected to other nodes. The partial mesh topology (shown in figure 2.3) ensures that information flow from one node to any other node is possible. A peer to peer topology uses a full mesh topology were every node in the network can act as a client and server [Copb]. Once a larger number of devices are connected in a topology then using a partial mesh topology or a peer to peer topology becomes more efficient than a star topology.

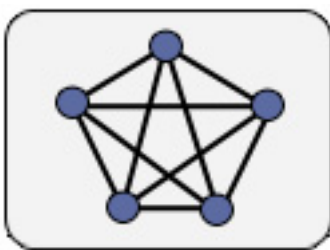


Figure 2.2: Mesh Topology

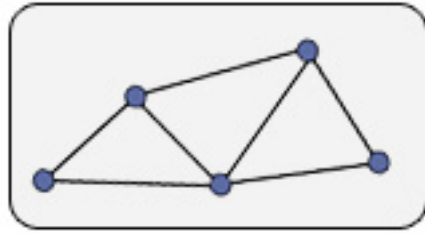


Figure 2.3: Partial Mesh Topology

2.5 Rule Engines: OO jDREW and Prova

OO jDREW [BC, BBH⁺05] is an Object Oriented extension to jDREW (java Deductive Reasoning Engine for the Web). OO jDREW extends jDREW by implementing order-sorted types, slots, and object identifiers of the RuleML language. RuleML has two different syntaxes the RuleML/XML syntax and the RuleML/POSL syntax and OO jDREW supports both syntaxes. OO jDREW has two main modes of operation: top-down and bottom-up. Bottom-up execution is used to infer all derivable knowledge from a set of clauses (forward reasoning). Top-down execution is used to solve a query on the knowledge base (backward reasoning). The other reasoning engine that will be discussed is Prova [KPS]. Prova is a rule engine that can execute declarative rules as well as reaction rules.

OO jDREW will be used as the centralized system used for the benchmarking between a centralized rule-based system and a distributed rule-based system. OO jDREW can be viewed as a distributed system with only one component and that component is the hub in a star topology. OO jDREW is the only contact point that the user will have with the centralized system; there will be no communication between other components. This allows centralized, hierarchical, and network systems to be all viewed as distributed architectures.

2.6 Rule Responder

Rule Responder [PBKC, Cra07, PBKC07, BP07, CB08] is a basis to build distributed intelligent rule-based applications for collaborative teams and virtual organizations. A virtual

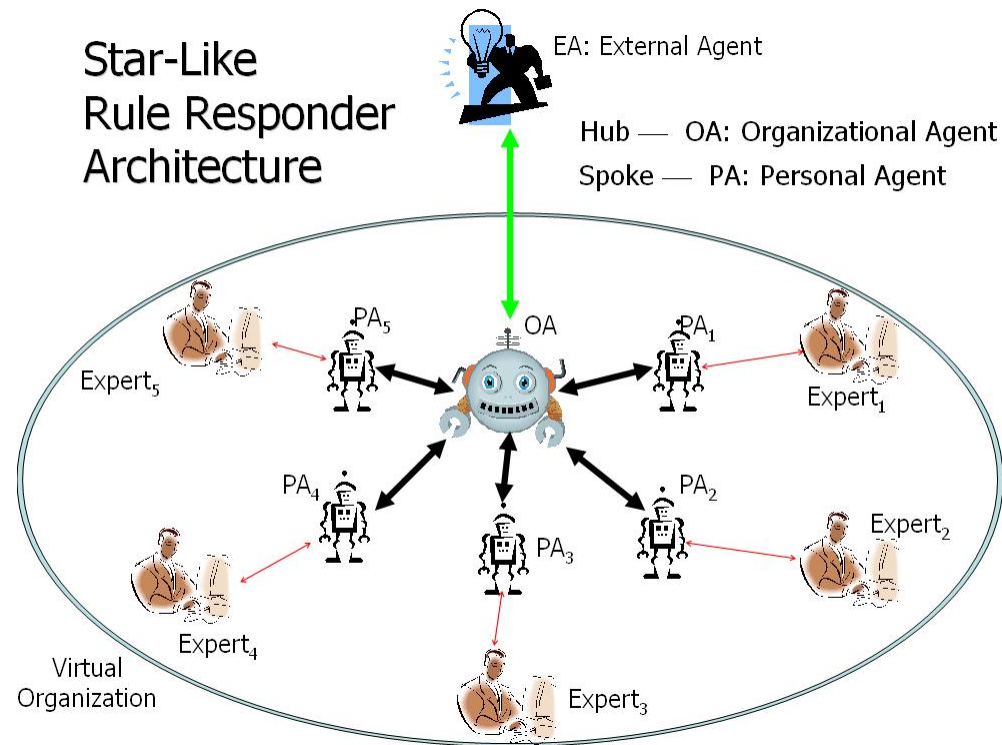


Figure 2.4: Rule Responder Architecture

organization is a group of people who collaborate over the internet and may not be part of the same real organization. Rule Responder uses RuleML as its XML-based knowledge interchange format. It is a multi-agent infrastructure that allows virtual organizations to collaborate in an automated manner. The agents communicated via reaction RuleML messages [PKB⁺]. Rule Responder is implemented as a Web Service architecture on top of Mule which is used to implement Rule Responder's hierarchical topology. Rule Responder has been used to realize a top-down query-answering service. It is currently implemented as a hierarchical distributed architecture but may be developed to use a networked distributed architecture. Rule Responder will be used as the hierarchical distributed rule-based system in the evaluation.

Rule Responder's architecture (shown in figure 2.4) presents the three different types of agents that realize Rule Responder. The external agents (EAs) represent a person outside of the virtual organization and that person wants to collaborate with the virtual organization. The organizational agent (OA) represents the organization as a whole and can be thought

as a mediator who delegates work to a particular personal agent (PA). Personal agents are assistants to a real member inside of the virtual organizational. The PAs contain rule bases that describe how the PA can assist the member of the virtual organization. OAs and PAs are realized through rule engines. OO jDREW [BC] is used to implement personal agents, while Prova [KPS] is used to implement organizational agents and some personal agents.

Evolving from a centralized to a hierarchical system involves connecting spokes (PAs) to a hub (OA) using a star topology. Currently, Rule Responder has support for a centralized hub (OA) and many spokes (PA) who communicate through the hub. All information must flow through this centralized hub in Rule Responder. Rule Responder is implemented with a star topology through the Mule ESB. Mule allows multiple configuration end points (spokes) and a centralized hub. This hierarchical system will gradually be extended to a networked system which will reduce communication overhead and eliminate bottleneck issues with the star topology.

Chapter 3

Knowledge Maintenance Issues in Centralized and Distributed Rule Systems

Rule Responder is implemented as a distributed rule-based system. It connects together organizational agents and personal agents so that they can share knowledge and external agents can query this knowledge. Each personal agent and organizational agents has their own set of rules and facts. The PA's knowledge corresponds to how their expert owner wants them to operate, while the OA's knowledge describes the virtual organization. All of the PAs contain knowledge bases that are stored at distributed locations. These distributed knowledge bases can also be described as a module. In contrast, a centralized rule system contains all of the knowledge in one centralized location and all of the knowledge would be saved as a single file or database.

3.1 Organization of Distributed Knowledge

Another view of Rule Responders distributed knowledge bases are as module-based logic programs [LM94]. Each module is a self contained rule base and in Rule Responder each

agent has their own module. Module logic programming is predicate centric while Rule Responder is person centric [LM94]. Person centric means that every fact is centered on a person while predicate centric focuses knowledge organization on the predicate name.

Person Centric:

```
phoneOf(ben, 1-506-270-3403)
emailOf(ben, ben.craig@unb.ca)
```

```
phoneOf(jim, 1-506-275-9712)
emailOf(jim, jim.lorde@unb.ca)
```

Predicate Centric:

```
phoneOf(ben, 1-506-270-3403)
phoneOf(jim, 1-506-275-9712)

emailOf(ben, ben.craig@unb.ca)
emailOf(jim, jim.lorde@unb.ca)
```

The first parameter of a person centric fact is a unique identifier (like a database key) to the person ‘Ben’ and ‘Jim’. The predicate centric method stores each ‘phoneOf’ fact together and each ‘emailOf’ fact together. Predicate centric and person centric concerns the top-level distinction between centralized and (all kinds of) distributed architectures. For centralized architectures issues of module boundaries and merging do not arise [LM94]. Only for distributed architectures do solutions need to be found here.

3.2 Module Boundaries

A major problem that arises with distributed logic programming is whether to allow backtracking across multiple modules which requires merging and crossing the boundaries of different modules [LM94]. For example if a query requested the phone number of all the people in the system then the query would have to backtrack across every module when using person centric modules. Another query may request all the information about a single

person which would require the query to retrieve information from a single module. A person centric query does not need to back track across multiple modules. It is possible to have queries that belong in both categories which sometimes require backtracking across multiple modules. However, back tracking across multiple modules is impossible for some systems.

In databases the order between facts (records) matter. The textual order is important in Prolog systems thus the in order in which facts and rule appear matters, so merging two modules into a single module is not trivial because it is impossible to preserve the textual order. In OO jDREW it is easy to merge two modules (union of them) because OO jDREW is more declarative then Prolog and does not rely on any textual order. So it is possible to back track across multiple modules using OO jDREW, but Rule Responder at this time does not require back tracking over multiple modules because it uses person centric modules and person centric queries.

3.3 Centralized Knowledge Maintenance

In a centralized rule-based system all knowledge is contained in a single location (either in a file or a database) and the knowledge is also stored in one rule format. For example the centralized system OO jDREW contains all the knowledge in the RuleML/POSL [Bol04] format. Having all the knowledge in one location allows a knowledge expert (or a knowledge team) to apply updates on the knowledge base. The knowledge expert also has complete control of what is contained within the knowledge base. Since the knowledge base is contained in single format (either RuleML/XML or RuleML/POSL for OO jDREW) then parsing the knowledge into the OO jDREW system requires no knowledge translations steps. When using a distributed system the knowledge format may not all be contained in a single rule language; so translations steps are required.

3.4 Distributed Knowledge Maintenance

In a distributed rule-based system there are several storage locations for the knowledge because each distributed agent requires a knowledge base to function. These agents are realized through a rule engine and each agent's rule engine may not be the same. Since the rule engine may be different then the knowledge base stored for that agent may be in a different rule format. To overcome this problem an interchange language is used so that agents can communicate with each other regardless of their rule engine implementation. The interchange language used by Rule Responder is Reaction RuleML [PKB07]. Reaction RuleML allows any two agents to communicate with each other. Another problem that is present with distributed rule-based systems is how to maintain the knowledge so that it is complete and consistent.

In a distributed rule-based system the problem of distributed knowledge maintenance is present. For centralized systems knowledge maintenance is simple, one person or a team looks after all of the knowledge stored in a single file or database. When using a distributed rule-based system the knowledge maintenance becomes more difficult. Since distributed systems allows each agent's owner to update their knowledge without ever being verified. It is possible that an agent may have inconsistent or incomplete knowledge. In order to prevent an inconsistent knowledge base each agent's knowledge must be verified through a set of consistency rules. These consistency rules [ZEF00] will ensure that their knowledge is complete and is consistent. This consistency check requires an extra processing step that a centralized rule-based system does not require. Without a consistency check the distributed rule-based system may run into unexpected errors.

Chapter 4

General Performance Issues of Distributed Topologies

There are several advantages of using a star topology in distributed systems and this chapter will evaluate the advantages but also explain the disadvantages of a star topology. A comparison of a P2P topology against a star topology will be presented. All distributed topologies have communication overhead, and deciding which topology to use depends on the communication overhead and performance of the topology. The distributed system must operate in an acceptable time for end users. An advantage of distributed systems over centralized systems is distributed processing which allows multiple computers to process a given problem. Rules execute faster when there are less clauses for the rule engine to process, so a distributed approach improves efficiency because there are multiple rule engines working on smaller knowledge bases instead of one rule engine working on a large knowledge base in a centralized approach. In a distributed rule system the knowledge is spread over many different physical locations and communication overhead may become a problem, but the communication overhead may not be noticeable to users.

4.1 Star Topology Advantages

The main advantage of a star topology is that every spoke is independent of every other spoke, so if one spoke is taking up a large amount of bandwidth then other spokes are not affected [Tho]. The isolation of spokes allows adding or removing of spokes from the hub to be simple. Increasing the scalability of the star topology is also quite easy because of the ease of adding new spokes. The isolation also prevents any broken spokes to not affect the rest of the topology. The only failure that will cause issues in the star topology is when the hub fails. The centralized aspect of the hub can permit the inspection of all traffic through the topology which allows for improved security and detection of suspicious behavior [Tho]. Trouble shooting is also much simpler in a star topology because all traffic goes through the centralized hub and each spoke can be tested individually. Another advantage of the star topology is that it is easy to understand and implement [Tho].

4.2 Star Topology Disadvantages

When a large amounts of traffic is flowing simultaneously through the star topology, the centralized hub can become overloaded [Tho]. In a P2P topology this does not happen because as more nodes are added there is more communication links that traffic can flow through. The star topology has a bottleneck issue that becomes present when large amounts of traffic are passing through the hub. When the hub becomes overflowed slowdowns occur in the topology. Another disadvantage with a star topology is that when the hub is broken then the entire system will go down, while in a P2P topology the remaining nodes would still function [Tho]. The dependency on the central hub causes the largest disadvantage of a star topology. The scalability, reliability and performance of the star topology rely on the centralized hub, but the simplicity of the topology makes it optimal for smaller distributed systems [Tho].

4.3 Advantages of Peer-to-Peer Networks

In a P2P topology all nodes can communicate with any other node. This difference allows a larger bandwidth limit for communication across the distributed system. Whenever a node is added to the topology the total computation and bandwidth capacity is increased [Copa]. The P2P topology increases the robustness of the distributed system because in case of failures a peer can be over taken by another peer and there are no visible faults in the distributed system. In P2P topologies there is not a single point of failure in the distributed system unlike the star topology [Copa].

4.4 Fault Tolerance of Distributed Topologies

When building a distributed system the system must be fault tolerant. This means that when a spoke or node fails in the system then the system is able to recover from the failure. In a star topology the isolation and centralization of spokes simplifies fault detection because whenever a spoke is broken then the centralized hub will know about it; however, if the centralized hub is broken then there is no way for the system to recover from this failure [Tho]. In a P2P topology problems of a single point of failure do not exist. If a peer goes offline another peer can act in place of that peer. Both topologies support a fault tolerance policy but the P2P topology has a more robust procedure for fault tolerance. The fault tolerance of a P2P system is much more complicated to implement then that of a star topology, so the star topology may still be the optimal choice for a distributed system [RGO07].

Chapter 5

Benchmarking Evaluation

The benchmarking was performed with the Rule Responder system (Distributed) and OO jDREW system (Centralized). The benchmarking was done using a use case creating for Rule Responder, and that use case is a symposium planner for the RuleML-2008 symposium [PBC]. The bench marking consisted of 5 queries in the OO jDREW system and the exact same queries in the Rule Responder system. The bench marking shows a qualitative analysis for the communication overhead in a distributed system and to verify if the communication overhead is an acceptable amount or not. The OO jDREW engine will consist of straight computation time, while the distributed system will consist of computation time for the personal agents, the organizational agent, and the communication time between the external agent, the organization agent and the personal agents.

5.1 Description of Use Case

One use case created to demonstrate Rule Responder is the organization of a symposium such as the RuleML-2008 Symposium, which is an example of a virtual organization that requires on line collaboration within a team [CB08]. Rule Responder can support the organizing committee of the RuleML-2008 Symposium by embodying responsibility assignment, automating first-level contacts for information regarding the symposium, helping the public-

ity chair with sponsoring, and screening of incoming submissions based on metadata (e.g., to see if the paper topics fit the symposium or not). It can also aid with other issues associated with the organization of a symposium, including presentation scheduling, room allocation, and special event planning.

The use case utilizes a single organizational agent to handle the filtering and delegation of incoming queries. Each committee chair has a personal agent that acts in a rule-constrained manner on behalf of the committee member. Each agent manages personal information, such as a Friend of a Friend (FOAF) profile containing a layer of personal information about the committee member as well as FOAF-extending rules. These rules allow the personal agent to automatically respond to requests concerning the RuleML-2008 Symposium. Task responsibility for the organization is currently managed through a responsibility matrix, which defines the tasks committee members are responsible for. The matrix and the roles assigned within the virtual organization are defined by an OWL (Ontology Web Language) Lite Ontology. The Pellet [HPSM] reasoner is used to infer subclasses and properties from the ontology.

External agents can communicate with the RuleML-2008 agents by sending messages that transport queries, answers, or complete rule sets to the public interface of the organizational agent (e.g., an HTTP port to which `post` and `get` requests can be sent from a Web form). The standard protocol for intra-transport of Reaction RuleML messages between Rule Responder agents is JMS. HTTP SOAP is used for communication with external agents, such as Web services or HTTP clients (Web browsers).

5.2 Centralized Performance (OO jDREW)

OO jDREW was tested using 5 personal agent rules bases from the RuleML-2008 use case. All of the modules were compiled into a single rule base and were executed using OO jDREW. The following table shows the computation time of the queries.

Query:	Computation Time (Milliseconds):
1) sponsor(contact[?Name,?Organization],5000:integer, results[?Level,?Benefits,?DeadlineResults], performative[?Action])	141
2) checkPendingPanelParticipants(?Meeting, ?Participant, ?Organization)	31
3) viewSponsors(?Meeting, ?Sponsor, ?SponsorLevel)	22
4) viewOrganizationPartners(?Meeting, ?Partner)	18
5) viewPanelTime(?Meeting, ?Time,?Day, ?Month, ?Year)	16

These results show that it does not take very long to compute the queries using OO jDREW. The most complex query to calculate is the first query and it took 141 milliseconds, the least complex query to calculate was the last query which took 16 milliseconds. This shows that a centralized system does not require much computation time to find the results to the queries.

5.3 Hierarchical Performance (Rule Responder)

The testing done for Rule Responder was done using the exact same 5 personal agents that OO jDREW used except for this time an organizational agent is required. Rule Responder requires an organizational agent so that queries can be delegated to the correct personal agent. Rule Responder is also a distributed system so communication overhead needs to be considered. The sequence diagram (figure 5.1) describes how Rule Responder communicates with the different agents in the RuleML-2008 use case.

Query(Queries are transcribed to POSL format would normally be in reaction RuleML):	Total Computation Time (Milliseconds):
sponsor(contact[?Name,?Organization],5000:integer, results[?Level,?Benefits,?DeadlineResults], performative[?Action])	3430

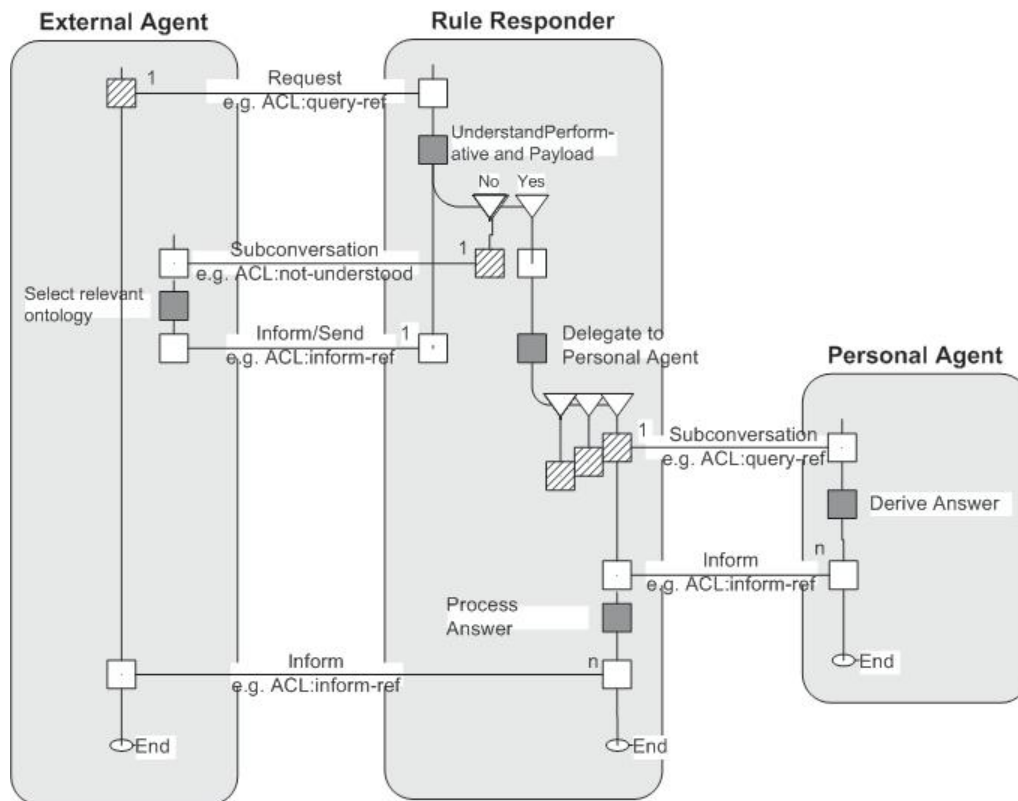


Figure 5.1: Communication Sequence for RuleML-2008 use case

<code>checkPendingPanelParticipants(?Meeting, ?Participant, ?Organization)</code>	4861
<code>viewSponsors(?Meeting, ?Sponsor, ?SponsorLevel)</code>	4057
<code>viewOrganizationPartners(?Meeting, ?Partner)</code>	9048
<code>viewPanelTime(?Meeting, ?Time,?Day, ?Month, ?Year)</code>	2780

The results using Rule Responder show a much higher execution time. This higher execution comes from the communication overhead of the distributed agents. For each query the EA requests; the query must be send to the OA. The OA then computes which PA will be able to solve the query. The OA will then send the query to the PA. The PA must solve the query and then send the answer(s) back to the OA. The OA can finally send the answer back to the EA. The results show that the computation time is only a small fraction of the total computation time. The variation in the query time is from the amount of solutions to the queries, the more results to the query then the longer the execution time because there are more messages being sent through the topology. The main concern when design a distributed rule-base system is the communication overhead that is required by the distributed system. A key design goal for a distributed system is to minimize the communication overhead [Arn].

5.4 Peer-to-Peer (Theoretical Efficiency Considerations)

Evolving the hierarchical system of Rule Responder to a networked system in the simplest form requires the communication between two personal agents [CB08]. An example of this peer to peer communication is possible, when (using the RuleML-2008 symposium use case as an example) the panel chair receives a wrong query which was actually meant for the publicity chair, then the panel chair could directly forward the query to the publicity chair. The Mule ESB has the capacity to evolve Rule Responder from a hierarchical system to a networked system. This networked system would save one communication step in the

communication process, because normally the query would have to be sent back to the OA and then the OA would send the query to the publicity chair. The panel chair could possibly CC the query to the OA so the OA would know that the publicity chair will be sending the OA an answer shortly. In both of these situations there is still reduction in the communication steps. Since minimization of communication overhead is a key design goal for distributed systems it is important to achieve as many speed ups as possible [Arn].

A peep to peer topology would allow PAs to send messages to each other [CB08]. Instead of having all communication going through the OA the PAs could communicate directly with other PAs. The P2P topology makes query delegation more complicated because not only OAs can delegate queries but also PAs. Each PA would need to know every other PA's addresses in a fully connected P2P topology or only a subset of PA's addresses in a partially connected P2P topology [Tea].

Once there are large amounts of information flowing simultaneously through a hierarchical rule-based system a networked system can be used to alleviate the bottleneck issues of the hierarchical system [Tea]. A P2P topology would theoretically perform better once there are so many agents communicating that a star topology bottlenecks [Copb]. Ignoring the bottleneck issue a star topology would still operate slower than a P2P topology, but the difference may not be noticeable [Tea]. There are less communication steps in a P2P topology because each message does not have to go through the centralized hub like in a star topology thus providing a speed up in communication overhead. However, a star topology is less complicated to implement, debug and maintain because all communication goes through the centralized hub. The maintenance of a P2P topology is more complicated, but can improve communication overhead because there are less communication steps in a P2P topology.

Chapter 6

Conclusion

When developing a rule system, a decision of creating the system as either centralized or distributed system must be decided. Once the decision for a distributed system has been decided, then a topology is required. Deciding which topology to use in a distributed system requires an analysis of the communication overhead, fault tolerance and performance issues of the topology. The different topologies discussed in the report showed the advantages and disadvantages of each topology. Selecting either a hierarchical or network topology is an important decision when building a distributed system and an extensive comparison between a hierarchical topology and a network topology has been presented. Developing a distributed rule-based system requires solutions to issues with distributed knowledge maintenance. The evaluation between OO jDREW (centralized) and Rule Responder (hierarchical) shows practical applications for centralized and distributed rule-based system, but also how centralized and distributed systems perform. A theoretical evaluation of a networked rule-based system has been discussed. The tools used in the evaluation: OO jDREW [BC], Prova [KPS], and Rule Responder [PBKC] are all open source and can be downloaded at the given citations.

Bibliography

- [Arn] Ken Arnold. Introduction to Distributed System Design. <http://code.google.com/edu/parallel/dsd-tutorial.html>.
- [BBH⁺05] Marcel Ball, Harold Boley, David Hirtle, Jing Mei, and Bruce Spencer. The OO jDREW reference implementation of ruleML. In Asaf Adi, Suzette Stoutenburg, and Said Tabet, editors, *RuleML*, volume 3791 of *Lecture Notes in Computer Science*, pages 218–223. Springer, 2005.
- [BC] Marcel Ball and Benjamin Craig. Object Oriented java Deductive Reasoning Engine for the Web. <http://www.jdrew.org/ooidrew/>.
- [BCC⁺] Antoine Borg, Travis Carlson, Alan Cassar, Andrew Cookeand, Stephen Fenech, and More. Mule. <http://mule.codehaus.org/display/MULE/Home>.
- [BGT05] Harold Boley, Benjamin Grosf, and Said Tabet. RuleML Tutorial. <http://www.ruleml.org/papers/tutorial-ruleml-20050513.html>, 2005.
- [Bol01] Harold Boley. Design Rationale of RuleML: A Markup Language for Semantic Web Rules. In *Semantic Web Working Symposium (SWWS'01)*, pages 381–401, July/August 2001.
- [Bol04] Harold Boley. POSL: An Integrated Positional-Slotted Language for Semantic Web Knowledge. <http://www.ruleml.org/submission/ruleml-shortation.html>, May 2004.
- [BP07] Harold Boley and Adrian Paschke. Expert querying and redirection with rule responder. In Anna V. Zhdanova, Lyndon J. B. Nixon, Malgorzata Mochol, and John G. Breslin, editors, *FEWS*, volume 290 of *CEUR Workshop Proceedings*, pages 9–22. CEUR-WS.org, 2007.
- [CB08] Benjamin Craig and Harold Boley. Personal Agents in the Rule Responder Architecture. In Nick Bassiliades, Guido Governatori, and Adrian Paschke, editors, *RuleML-2008*, Lecture Notes in Computer Science. Springer, 2008.
- [Copa] James Cope. Advantages of peer-to-peer networks. <http://www.solyrich.com/p2p-pros-cons.asp>.
- [Copb] James Cope. QuickStudy: Peer-to-Peer Network. <http://www.computerworld.com/action/article.do?command=viewArticleBasicarticleId=69883>

- [Cra07] Benjamin Craig. The OO jDREW Engine of Rule Responder: Naf Hornlog RuleML Query Answering. In Adrian Paschke and Yevgen Biletskiy, editors, *RuleML-2007*, volume 4824 of *Lecture Notes in Computer Science*. Springer, 2007.
- [HPSM] James Hendler, Bijan Parsia, Evren Sirin, and More. Pellet: The Open Source OWL DL Reasoner. <http://pellet.owldl.com/>.
- [KPS] Alex Kozlenkov, Adrian Paschke, and Michael Schroeder. Prova: A Language for Rule Based Java Scripting, Information Integration, and Agent Programming. <http://www.prova.ws/>.
- [LM94] Evelina Lamma and Paola Mello. Modularity in logic programming. In *Proceedings of the eleventh international conference on Logic programming*, pages 15–17, Cambridge, MA, USA, 1994. MIT Press.
- [PBC] Adrian Paschke, Harold Boley, and Ben Craig. RuleML-2008 Use Case. <http://www.ruleml.org/RuleML-2008/RuleResponder/index.html>.
- [PBKC] Adrian Paschke, Harold Boley, Alexander Kozlenkov, and Benjamin Craig. Rule Responder: A RuleML-Based Pragmatic Agent Web for Collaborative Teams and Virtual Organizations. <http://www.responder.ruleml.org>.
- [PBKC07] Adrian Paschke, Harold Boley, Alexander Kozlenkov, and Benjamin Craig. Rule Responder: RuleML-Based Agents for Distributed Collaboration on the Pragmatic Web. In *2nd ACM Pragmatic Web Conference 2007*. ACM, 2007.
- [PKB⁺] Adrian Paschke, Alexander Kozlenkov, Harold Boley, Michael Kifer, Said Tabet, Mike Dean, and Keara Barrett. Reaction RuleML. <http://ibis.in.tum.de/research/ReactionRuleML/>.
- [PKB07] Adrian Paschke, Alexander Kozlenkov, and Harold Boley. A Homogenous Reaction Rule Language for Complex Event Processing. In *Proc. 2nd International Workshop on Event Drive Architecture and Event Processing Systems (EDA-PS 2007)*. Vienna, Austria, September 2007.
- [RGO07] Arnon Rotem-Gal-Oz. Advantages of peer-to-peer networks, 2007.
- [Tea] The Learn-Networking.com Team. A Guide to Network Topology. <http://learn-networking.com/network-design/a-guide-to-network-topology>.
- [Tho] Karl Thomas. Advantages of peer-to-peer networks. <http://fallsconnect.com/topology.htm>.
- [TvS] Andrew S. Tanenbaum and Maarten van Steen. Distributed Systems: Principles and Paradigms. <http://www.cs.vu.nl/~ast/books/ds1/>.
- [ZEF00] Andrea Zisman, Wolfgang Emmerich, and Anthony Finkelstein. Using xml to build consistency rules for distributed specifications. In *IWSSD '00: Proceedings of the 10th International Workshop on Software Specification and Design*, page 141, Washington, DC, USA, 2000. IEEE Computer Society.