Chapter 5
Argumentation-Enabled Web-Based Intelligent Decision Support System (Web@IDSS)

5.1 Introduction

In any enterprise, information is one of the essential components required for decision making. Traditional information systems have been used by enterprises to consider the underlying information of an enterprise and assist them in this process. However, these systems are basic and are inflexible in responding to current situations such as:

- Dealing with the huge increase of information. In recent years, there has been a huge increase in the amount of information available, termed the tsunami of data (Brodie 2008a, b). In order to make informed decisions, enterprises may have to consider a huge volume of information as that may contain hidden informed knowledge. So, information systems need to process this information autonomously and make it available to decision makers to assist them in the decision-making process.
- Dealing with information that may be across and beyond an enterprise's boundaries. For example, in the context of Customer Relation Management (CRM) software, for the development of new products, considering information such as expert knowledge, customer opinions, reviews about existing products and services etc. in the decision-making process may lead to better results.

In order to overcome these issues, Decision Support Systems (DSS) (such as individual DSS and Group DSS) were developed that assist in a wide range of enterprise-wide decision-making processes (Power 2002; Power and Sharda 2009). To consider the multi-site nature of decision making due to the widespread adoption of the WWW, Web-based DSS (Yao et al. 2001) were developed by which decision makers that are spread across different locations can collaborate in the decision-making process (Vahidov and Kersten 2004; Silverman et al. 2001; Toni 2007). By using Semantic Web technologies, Web-based DSS, with help of ontologies, can understand and consume information which exists outside an enterprise's boundaries. The challenge that now confronts the current Web-based DSS systems is: *how to take into account the information exists within an enterprise and/or in other enterprises that may*

N. K. Janjua, *A Defeasible Logic Programming-Based Framework to Support Argumentation in Semantic Web Applications*, Springer Theses,
DOI: 10.1007/978-3-319-03949-7_5, © Springer International Publishing Switzerland 2014

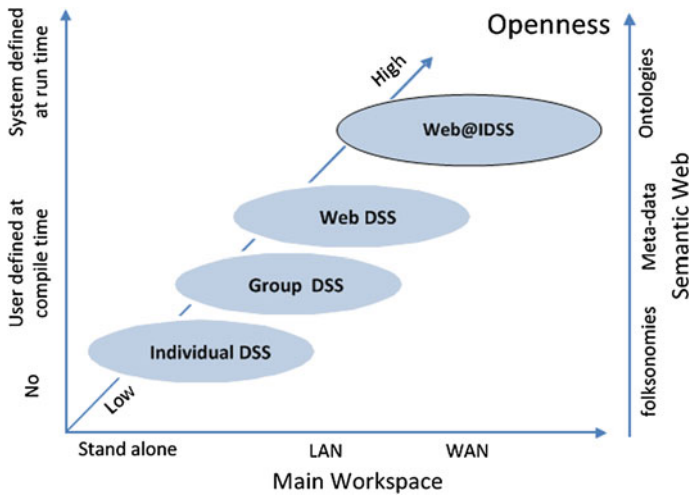

Fig. 5.1 Evolution towards Argumentation-enabled Web-based IDSS (extended from Lee and Chung 2005)

be potentially incomplete and/or contradictory (within themselves and/or with the existing information in an enterprise) and utilize it in their decision-making process.

To address this challenge, as mentioned in Sect. 3.3, Web-based DSS have been developed that are based on defeasible reasoning to represent incomplete information and reason using pre-defined preferences from a single user's point of view to resolve conflicts (Antoniou and Bikakis 2007; Bassiliades et al. 2004; Grosof et al. 2002). However, these systems fail to address the problem in the context when information may come from different sources such as in group decision making where there is more than one decision maker involved in the decision-making process where conflicts may arise between the members of the group due to their different viewpoints. To address this challenge, I propose a framework for an Argumentation-enabled Web-based Intelligent DSS (Web@IDSS). The proposed framework will use logic-based language for information representation and argumentation-driven reasoning to identify and resolve conflicts in the information coming from different sources, followed by information integration to assist a decision maker in his decision-making process. This will advance the research in Web-based DSS as depicted in Fig. 5.1.

The organization of this chapter is as follows: in Sect. 5.2, the problem to be addressed is outlined by using a case study that highlights the requirements and challenges for Web-based DSS in an enterprise. In Sect. 5.3, an overview of the proposed framework for Argumentation-enabled Web-based Intelligent DSS (Web@IDSS) is given. From Sects. 5.4 to 5.6, each component of the proposed framework is explained in detail and the ways in which it provides a solution to the problem highlighted in the case study is discussed. Section 5.7 concludes the chapter.

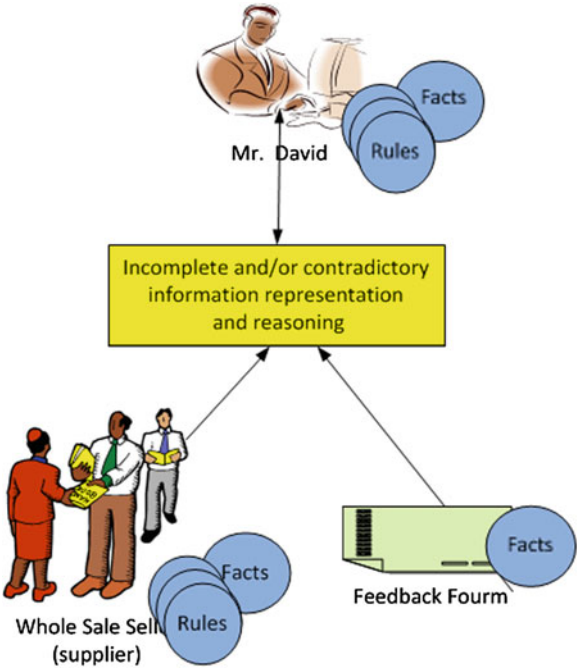

Fig. 5.2 Analyses of the business policies of a supplier and feedback provided by the other users (companies) by Mr. David

5.2 Case Study for Problem Definition

To explain the problem with an example, consider a scenario where Mr David is a marketing manager of an enterprise A. He is responsible for formulating and suggesting business strategies to increase sales of the company's products (existing and new) and generate revenue for the Chief Executive Officer (CEO) of the enterprise A. Enterprise A intends to manufacture a new product (say Product B). To increase the enterprise's revenue from this project, one of the important aspects that Mr. David identifies is "the greater the discount that an enterprise A receives from the supplier, the cheaper the new product", and negotiation plays an important part in securing the maximum discount. Mr. David identifies that the materials for manufacturing the product will be sourced from 'N' different suppliers, each of whom offer varying levels of discount. Mr David would like to select a supplier that may give his enterprise the maximum discount and he needs to justify his selection to the CEO of the company.

To achieve his objective, Mr David needs to analyse the business policies of each supplier against his company's requirements along with the feedback provided by the other users (companies) about the raw materials provided by the suppliers as shown in Fig. 5.2. During this process, Mr. David will come across different challenging situations such as follows:

- There might be conflicts between the supplier's policies and an enterprise A's business requirements.
- There may be conflicts within the supplier's business policies.
- A situation may arise where Mr. David may require some information for decision making which is not available at the time of decision making.

In order to overcome the above mentioned challenging situations, Mr. David requires a Web-based DSS that will assist him to overcome these challenges. The Web-based DSS should have the following functionalities:

(a) an interface to define the requirements in the form of business rules such as 'Purchase product from supplier only if product feedback is good' and certain facts or information to realize those rules;
(b) an interface to download the supplier's product information and public policies with details on the possible discount that can be given on their products and services;
(c) the capability to download feedback or reviews from other users (companies) on the suppliers' products from a third party forum such as Amazon;
(d) situations may arise where the business policies of a supplier may be incomplete or negotiation is required between the supplier and an enterprise A to resolve conflicting interests. The Web-based DSS should be able to cater for these and provide a means of resolving these conflicts, with a justified explanation, during the reasoning process;
(e) the capability to provide a graphical representation of the reasoning process and the result in order to make them easily understandable by non-technical persons such as CEOs.

To have such functionalities, a Web-based IDSS is needed that is able to capture the information outside an enterprise's boundaries, identify the goals, identify any conflicts in the information with respect to the goals, resolve these conflicts by reasoning over them and show the basis of the reasoning by which a conclusion is reached. The current Web-based DSS are not able to represent, reason and integrate the information that is required for the abovementioned tasks. Therefore, to address this challenge, Mr. David's requirements, which should be incorporated in Web-based IDSS, are formalized as follows:

- A declarative, logic-based language for specification of the business requirements of an enterprise.
- The declarative language should have the capability to represent incomplete and contradictory information (i.e. business rules and facts).
- An inference mechanism that can perform reasoning pertaining to incomplete and/or contradictory information in the knowledge base.
- Graphical representation of results obtained from the reasoning process to assist in decision making.
- Justifiable explanation of the results obtained after the reasoning and conflict resolution has occurred.

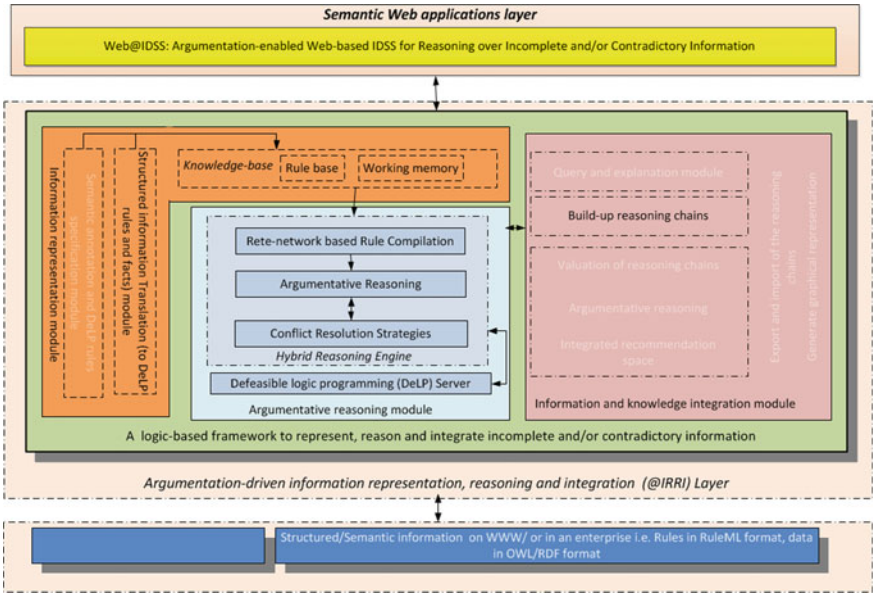

Fig. 5.3 Proposed conceptual framework with highlighted components exploited by Web@IDSS

Assumption

- Enterprise A, the supplier and the feedback forum share a common vocabulary defined in RDF/XML format and the predicates defined in the vocabulary are used for the specification of business rules and policies. Therefore, the information taken into account by the Web-based IDSS is structured information.

To achieve the abovementioned objectives, in the next section, a Web@IDSS framework is proposed that can represent, reason and integrate incomplete and/or contradictory information which exists within an enterprise and/or in other enterprises to assist the decision maker in the decision-making process.

5.3 Proposed Framework for Argumentation-Enabled Web-Based IDSS (Web@IDSS)

In this section, the solution for an Argumentation-enabled Web-based IDSS is proposed to represent, reason and integrate incomplete and/or contradictory information exits within an enterprise and/or in other enterprises. Figure 5.3 represents the proposed framework and consists of three layers as follows:

1. Information layer
   The information layer represents the structured information identified by the decision maker to be considered during the decision-making process. This information may include:
   - Business policies of an enterprise that provides different products and services published on the WWW.
   - Feedback of users published on the WWW about the products and services offered by an enterprise.
2. @IRRI layer
   This layer comprises a logic-based framework that enables a Web-based DSS to deal with information which is potentially incomplete and/or contradictory, and to process and consider it for decision making. It provides different modules to represent or translate the information into DeLP format, perform hybrid reasoning for arguments construction from underlying information followed by conflicts resolution and then integrate the information obtained from the hybrid reasoning to assist the decision maker in the decision-making process. The modules are as follows:
   (a) *The Information representation module* is responsible for
      - the pre-processing of potentially incomplete and/or contradictory information, and
      - the translation of pre-processed information to DeLP format and saving it in the knowledge base.

   (b) *The Argumentative reasoning module* performs hybrid reasoning over information saved in the knowledge base. The hybrid reasoning engine performs two types of reasoning such as:
      - data-driven reasoning for arguments construction, and
      - goal-driven reasoning for conflicts identification followed by their resolution.

   (c) *The Information and knowledge integration module* is responsible for
      - performing integration of the output of the hybrid reasoning in the form of a reasoning chain;
      - categorization of the reasoning chains based on the types of arguments they are built on;
      - graphical representation of the reasoning chain.
3. Web-based decision support systems (Web-based DSS) layer
   This layer consists of Web-based DSS such as Web@IDSS, that exploits the @IRRI layer and the information layer to achieve its objectives.

Before explaining the working of the proposed framework, in the next sub-section, several important definitions and concepts are introduced that are pivotal to understand the working of the proposed framework for Web@IDSS.

5.3.1 Important Definitions

In this section, the important concepts that encompass *syntax and semantics* for DeLP to make it suitable for information representation, reasoning and integration in Semantic Web applications are defined as follows:

5.3.1.1 DeLP Language

DeLP language is a set Φ containing a set of predicates $\mathcal{P}$, a set of functions $\mathcal{F}$, an infinite set of variables $\mathcal{X}$, a finite set of symbols $\mathcal{S}$, and a set of labels $\mathcal{L}$. Mathematically, language is defined as follows:

$$\Phi = \{\mathcal{P}, \mathcal{F}, \mathcal{X}, \mathcal{L}, \mathcal{S}\} \tag{5.1}$$

The language supports two types of negation: *strong negation*, represented by the symbol $\sim \in \mathcal{S}$ to represent contradictory knowledge, and *weak negation* which represents negation as failure represented by the symbol *not* $\in \mathcal{S}$ which is used to represent incomplete information.

5.3.1.2 Working Memory

A collection of facts is known as working memory. Considering a set $\mathcal{P}$ of predicates and an infinite set of variables $\mathcal{X}$, a fact is a ground predicate $f \in \mathcal{P}$, or a negated ground predicate $\sim f \in \mathcal{P}$. A set of facts, i.e. working memory is represented by $\mathcal{WM}$. Mathematically, working memory is defined as follows:

$$\mathcal{WM} = \{f \cup \sim f \mid f, \sim f \text{ are ground predicates}\} \tag{5.2}$$

where a ground predicate is a predicate whose input arguments are constant. The predicate p(a, b) and *not* p(a, b) are ground predicates. Facts represent the current state of the world and these provide some sort of evidence as a basis for activating the rules of inference to infer new facts. If there are no facts in the system, then no inference rules will be activated.

5.3.1.3 Production Rule

Production rules are rules of the form *IF Condition DO Action*, where Condition queries the working memory containing the facts on which the rules operate. Formally, a production rule $\mathcal{A}$ is of the form: [rule identifier] [rule body] [type of inference rule] [conclusion]. Mathematically, a production rule is defined as follows:

$$[\mathcal{A}]\, \nabla \vdash \alpha \tag{5.3}$$

where

- [rule identifier]: $\mathcal{A} \in \mathcal{L}$ is used as the identifier or name of the production rule;
- [rule body] ∇ is a pattern in the body of a production rule $\mathcal{A}$. A pattern is a tuple of predicates i.e. $\nabla \subseteq \mathcal{P}$, and defined as $\nabla = (C_i, \ldots, C_j)$ where $0 < i < j$, C_i is a predicate in a pattern;
- [conclusion] α is a predicate whose instances could be intuitively considered to be added to the working memory when the rule is fired during argument construction defined later on; and
- [type of inference] $\vdash$ indicates the inference that associates the rule body with the conclusion.

The production rule represents a reasoning step for α from a tuple of predicates $\{C_1, \ldots, C_n\}$. The language supports two types of inferences in production rules. One is strict inference represented by the symbol $\rightarrow \in \mathcal{S}$ and the second is defeasible inference represented by the symbol $\dashrightarrow \in \mathcal{S}$. Strict inference is used to represent information about which there is no ambiguity, whereas defeasible inference is used to represent ambiguous or tentative information. Strong negation is allowed at the conclusion of the rule, whereas weak negation is allowed only in the body of the rule.

5.3.1.4 Rule Base

The set of production rules is known as the rule base, denoted by $\mathcal{R}$. Mathematically, the rule base is defined as follows:

$$\mathcal{R} = \{production\ rule\} \tag{5.4}$$

5.3.1.5 Strict Production Rule

Strict production rules are the rules in the classical sense: when a rule's conditions are true, apply the rules and reach a conclusion. These rules are used to represent an inference mechanism from conditions to conclusion without any doubt. Most of the time, these rules are constructed from terms such as 'should be', 'must be', 'must' and their opposite terms. Formally, a production rule $\mathcal{S} \in \mathcal{R}$ is a strict production rule of the following form if the rule is based on strict inference.

$$[\mathcal{S}]\, \nabla \rightarrow \alpha \tag{5.5}$$

The strict production rule $\mathcal{S} \in \mathcal{R}$ is used to represent truthful information which contains no ambiguity. Consider rule r1 which states that 'if a person is innocent and has no crime history then he is not guilty' and rule r2 which states that 'if someone is not guilty, then he is free'. These rules can be represented as strict production rules thus:

- $[r1]innocent(X), hasCrimeHistory(X, no) \rightarrow\sim guilty(X)$
- $[r2]not\ guilty(X) \rightarrow free(X)$.

5.3.1.6 Defeasible Production Rule

Defeasible rules or refutable rules are those that link the set of conditions to a conclusion with a certain doubt, and therefore could be refuted by contrary evidence. This type of rule is indicated by words like 'usually', 'presumably', or 'sufficiently' or we could intuitively feel that it is refutable. Formally, a production rule $\mathcal{D} \in \mathcal{R}$ is a defeasible production rule of the following form:

$$[\mathcal{D}]\, \nabla \dashrightarrow \alpha \tag{5.6}$$

A defeasible production rule $\mathcal{D} \in \mathcal{R}$ is used to represent tentative information which may change in due course. Consider rule r3 that states: 'assume that someone is innocent whenever it has not been proven that he is guilty' and rule r4 that states: 'generally, do not cross the railway tracks if it cannot be proven that no train is coming'. These rules can be represented as defeasible production rules as follows:

- $[r3]not\ guilty(X) \dashrightarrow innocent(X)$.
- $[r4]not\ \sim train\_is\_coming \dashrightarrow\sim cross\_railway\_tracks(X)$.

5.3.1.7 Argumentative Production System

An argumentative production system is defined as a system that allows representation and execution (i.e. reasoning) of both strict and defeasible production rules. It consists of a knowledge base (i.e. consisting of working memory and a rule base) and a hybrid reasoning engine. An argumentative production system is formally defined as follows:

$$\mathbb{P} = (\mathcal{WM}, \mathcal{R}, Args) \tag{5.7}$$

- where $\mathbb{P} \in \mathcal{L}$ is a label to identify the argumentative production system.
- $\mathcal{WM}$ represents the initial collection of facts in the argumentative production system.
- $\mathcal{R}$ is the set of rules comprising both strict and defeasible production rules in the argumentative production system.
- $Args$ is an active argument set which contains arguments generated during the argument construction phase, which will be defined later. Prior to the argument construction phase, the $Args$ is an empty set.

5.3.1.8 Consistency

A set of rules is consistent if and only if there are no two rules with mutually contradictory predicates as their conclusion. Mathematically, this is represented as follows:

$$\mathcal{R}_{consis} = \{\forall r, s \in \mathcal{R} | \, if \; r \vdash \alpha \, then \; s \nvdash \sim \alpha\} \tag{5.8}$$

5.3.1.9 Arguments Construction

Arguments construction is defined as a recursive process which involves the interpretation of production rules with function match $(\mathcal{WM}, \mathcal{R}) \in \mathcal{F}$ which looks for rules from a rule base whose pattern matches the facts in $\mathcal{WM}$ and, on a successful match, executes the production rule which then adds the rule's conclusion i.e. ground predicate, to the working memory and instance of the production rule i.e. argument, to the argument set i.e. $Args$. Such a reasoning process is also known as data-driven reasoning. The argument construction process continues until all the matched rules in the knowledge base have been processed. This interpretation of a production rule is also known as the 'firing of a rule'.

$$\begin{aligned} &\forall \, r \in \mathcal{R}\{\nabla \in r, \alpha \in r, r \notin Args \mid if \; match(\nabla, \mathcal{WM}) \\ &then \; \mathcal{WM}\prime = \mathcal{WM} \cup \alpha' \; and \; Args = Args \cup r'\} \end{aligned} \tag{5.9}$$

where α' is the ground predicate and r' is the interpreted rule by function *match* $(\mathcal{WM}, \mathcal{R}) \in \mathcal{F}$. The $Args$ contains interpreted rules or fired rules known as arguments.

5.3.1.10 Strict Argument

A fired production rule in an argument set with strict inference is called a 'strict argument'. Mathematically, this is represented as follows:

$$[\mathcal{S}] \; \beta_1, \ldots, \beta_n \rightarrow \alpha \tag{5.10}$$

where

1. $\mathcal{S} \in \mathcal{L}$ is the label of the argument
2. α is a ground predicate known as the 'claim of an argument'. Function $claim(\mathcal{S})$ $\in \mathcal{F}$ returns the claim of a given argument $\mathcal{S}$.
3. β_i is a ground predicate known as the premise of an argument, supporting the claim of an argument. Function $premises(\mathcal{S})$ returns a set of argument premises $\mathcal{S}$.
4. $\rightarrow$ represents a strict inference from the set of premises to the claim.

5.3.1.11 Defeasible Argument

A fired production rule in an argument set with defeasible inference is called a 'defeasible argument'. Mathematically, this is represented as follows:

$$[\mathcal{D}]\beta_1, \ldots, \beta_n \dashrightarrow \alpha \tag{5.11}$$

where

1. $\mathcal{D} \in \mathcal{L}$ is the label of an argument.
2. α is a ground predicate known as the 'claim of an argument'. Function $claim(\mathcal{D}) \in \mathcal{F}$ returns the claim of a given argument $\mathcal{D}$.
3. β_i is a ground predicate known as the premise of an argument, supporting the claim of an argument. Function $premises(\mathcal{D})$ returns a set of argument premises $\mathcal{D}$.
4. $\dashrightarrow$ represents defeasible inference from the set of premises to the claim.

To avoid any fallacies in the argumentation process, the following restrictions on strict and defeasible argument structure are considered:

1. A premise in an argument cannot simultaneously be a conclusion i.e. $\beta_i \notin \alpha$.
2. A negation of a claim cannot become the premise of a claim i.e. $\beta_i \neq\sim \alpha$.
3. There is no redundancy of a premise in a pattern. $\beta_i \neq \beta_j$ where $1 < i, j <$ n.

5.3.1.12 Counter-Argument

An argument r counter-argues argument s if and only if $claim(r)$ is inconsistent with $claim(s)$ or $claim(r)$ is inconsistent with the $premises(s)$. Mathematically, a counter-argument is defined as :

$$\forall r, s \; \{if\,(!Consistent(claim(s),\ claim(r)))\ then\ r \diamondsuit s\} \tag{5.12}$$

where $\diamondsuit$ is used to represent the counter-argument relationship between two arguments.

If argument r counter-argues argument s such that $claim(r)$ is inconsistent with $claim(s)$, it is called a 'direct counter-argument', and if argument r counter-argues s such that $claim(r)$ is inconsistent with $premises(s)$, then it is called an 'indirect counter-argument'. Mathematically, direct and indirect counter-arguments are represented as follows:

$$\forall s, r\{if\ !Consistent(claim(s), claim(r))\ then\ s \diamondsuit_{direct}\ r\} \tag{5.13}$$

$$\forall s, r\{if\ !Consistent(claim(s), premises(r))\ then\ s \diamondsuit_{indirect}\ r\} \tag{5.14}$$

A strict rule cannot counter-argue another strict rule because of the definition of consistency.

5.3.1.13 Static Defeat

Under certain conditions, an argument r defeats its counter-argument s by establishing its priority over its counter-argument. Such defeat is known as a 'static defeat'. The conditions for static defeat are as follows:

- If a strict argument counter-argues a defeasible argument, the strict argument always defeats a defeasible argument. In other words, the strict argument has higher priority than the defeasible argument. Mathematically, this is represented as follows:

$$\forall d, s \in Args\{if\ s,\ d\ are\ strict\ and\ defeasible\ arguments,\ respectively \mid s \diamondsuit_{direct} d \quad then\ s > d\} \tag{5.15}$$

- If a defeasible argument directly counter-argues a strict argument, then the strict argument defeats the defeasible argument. Mathematically, this is represented as follows:

$$\forall s, d \in Args\{if\ s, d\ \ are\ strict\ and\ defeasible\ \ arguments,\ \ respectively \mid d \diamondsuit_{direct} s \quad then\ s > d\} \tag{5.16}$$

5.3.1.14 Dialectical Tree

If an argument $\mathcal{A}$ counter-argues argument $\mathcal{B}$, and no static defeat exists, then a dialectical tree (as defined by Garcia and Simari 2004) for argument $\mathcal{A}$ is constructed to determine whether argument $\mathcal{A}$ defeats argument $\mathcal{B}$ or vice versa.

Let $\mathcal{A}$ be an argument. A dialectical tree for argument $\mathcal{A}$ is $\Sigma(\mathcal{A}, \mathrm{h})$ where h is claim($\mathcal{A}$), is recursively defined as follows:

(1) A single node labeled with an argument ($\mathcal{A}$, h) with no counter-argument is by itself a dialectical tree for ($\mathcal{A}$, h). This node is also the root of the tree.
(2) Suppose that $\Sigma(\mathcal{A}, \mathrm{h})$ is an argument with counter-arguments ($\mathcal{A}_1$, h_1), ($\mathcal{A}_2$, h_2),…,($\mathcal{A}_n$, h_n), the dialectical tree for ($\mathcal{A}$, h), $\Sigma(\mathcal{A}, \mathrm{h})$ is constructed by labeling the root node with ($\mathcal{A}$, h) and by making this node the parent of the root of dialectical trees for ($\mathcal{A}_1$, h_1), ($\mathcal{A}_2$, h_2),…,($\mathcal{A}_n$, h_n) i.e. $\Sigma(\mathcal{A}_1, \mathrm{h}_1)$, $\Sigma(\mathcal{A}_2, \mathrm{h}_2)$,…, $\Sigma(\mathcal{A}_n, \mathrm{h}_n)$. Figure 5.4 depicts the graphical representation of the dialectical tree.

5.3.1.15 Marking of Dialectical Tree

To identify the priority between an argument and its counter-argument, the dialectical tree is marked as either *defeated or undefeated* as shown in Fig. 5.5. If the dialectical

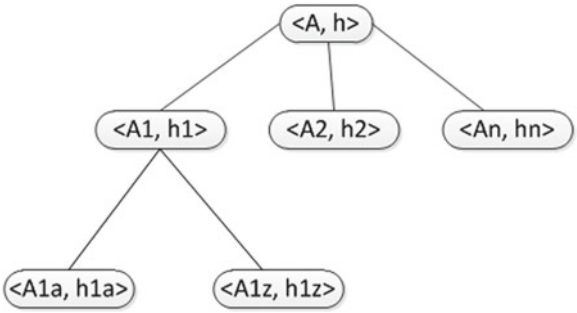

Fig. 5.4 Pictorial representation of a dialectical tree

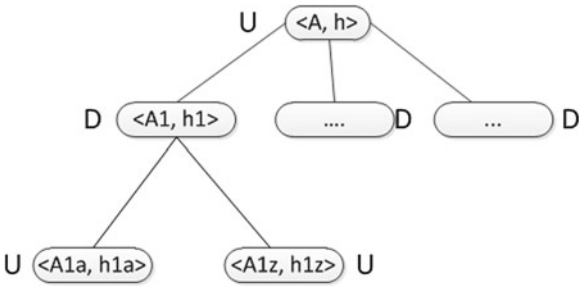

Fig. 5.5 Pictorial representation of a marked dialectical tree

tree of an argument is marked defeated, then the argument has less priority over its counter-argument and vice versa. The marking of the dialectical tree (as defined by Garcia and Simari 2004) is a two-step process as follows:

(1) Leaves of $\Sigma(\mathcal{A}, h)$ are U-nodes.
(2) Let $(\mathcal{B}, q)$ be an inner node of $\Sigma(\mathcal{A}, h)$. Then (B, q) will be a U-node iff every child of $(\mathcal{B}, q)$ is a D-node. The node $(\mathcal{B}, q)$ will be a D-node if it has at least one U-node as a child.

5.3.1.16 Dynamic Defeat

If an argument *r* counter-argues argument *s* and no static defeat exits, then dynamic defeat is computed. Let $\Sigma_U(\mathcal{A}, h)$ be marked dialectical tree for argument $\mathcal{A}$ and $\Sigma_D(\mathcal{B}, \sim h)$ is marked dialectical tree for its counter-argument $\mathcal{B}$, then argument $\mathcal{A}$ establishes its priority over its counter-argument $\mathcal{B}$ known as dynamic defeat. The dynamic defeat results in the establishment of the priority of an argument over its counter-argument which is known as a dynamic priority. Mathematically, dynamic priority is defined as follows:

$$\forall r, s \in Args\{if\ r \Diamond s, \Sigma_U(r, h), \Sigma_D(s, \sim h)\}\ then\ r > s \tag{5.17}$$

If an argument $\mathcal{A}$ has an undefeated dialectical tree i.e. $\Sigma_U(\mathcal{A}, h)$ and it counter-argues an argument $\mathcal{B}$ which also has an undefeated dialectical tree i.e. $\Sigma_U(\mathcal{B}, \backsim h)$, then neither argument A nor B can establish its priority over the other, resulting in a blocked situation. Such arguments are referred to as blocking arguments.

5.3.1.17 Sub-Argument

Given an argument set $Args$, an argument s is a sub-argument of r if and only if claim(s) $\subseteq$ $premise(r)$ and, if there exists say, counter-argument g, then the marked dialectical tree of an argument s is undefeated and the marked dialectical tree of argument g is defeated. Mathematically, the condition for a sub-argument can be represented as follows:

$$\forall r, s, g\{if\,(claim(s) \subseteq premise(r)\ and\ \ if\,(s \Diamond g)\ then\ s > g)\ then\ s\ \xi\ r\ \} \tag{5.18}$$

where ξ used to represent the sub-argument relationship between two arguments.

The sub-argument is a supporting argument and it must have the following characteristics:

1. argument s is consistent w.r.t argument r;
2. There is no $premise(s)$ such that $premise(s) \subseteq claim(r)$.

A sub-argument that provides support to another argument results in a chaining of arguments.

5.3.1.18 Reasoning Chain

An argument $\mathcal{A}$ supported by a chain of sub-arguments produces a reasoning chain $\lambda_{\mathcal{A}} = (\mathcal{A}_1, \ldots, \mathcal{A}_n)$ for an argument $\mathcal{A}$. The claim of supported argument $\mathcal{A}$, is called a 'result' of the reasoning chain and the chain of sub-arguments is called a 'support' for the result of the reasoning chain. Mathematically, a reasoning chain is defined as follows:

$$\forall r,\ s \in Args\ \{if\,(s\xi r)\ then\ \lambda_{(r,j)} = \lambda_{(r,j)} \cup\ s \tag{5.19}$$

where ξ is used to represent a sub-argument relationship and $\lambda_{(r,j)}$ is used to represent a reasoning chain with result j. The reasoning chain should have the following characteristics:

1. The reasoning chain is consistent (i.e., there is no contradiction in the result and support for the result).
2. There is no defeated argument in a reasoning chain.
3. Two blocking arguments cannot be in the same reasoning chain.

5.3.1.19 Strict Reasoning Chain

A reasoning chain is considered to be strict if all the arguments in the reasoning chain are strict arguments. Mathematically, a strict reasoning chain can be represented as follows:

$$\forall r, s \in \lambda_{(r,j)}\{r, s \textit{ are strict arguments}\} \tag{5.20}$$

This reasoning chain cannot be directly counter-argued by other reasoning chains. However, this reasoning chain can counter-argue and defeat the rest of the reasoning chains in an argumentative production system.

5.3.1.20 Defeasible Reasoning Chain

A reasoning chain is a defeasible reasoning chain if all arguments in the reasoning chain are defeasible arguments. Mathematically, defeasible reasoning chains can be represented as follows:

$$\forall d, f \in \lambda_{(r,j)}\{d, f \textit{ are defeasible arguments}\} \tag{5.21}$$

This reasoning chain can counter-argue or can be counter-argued by other reasoning chains in an argumentative production system. The defeasible arguments must be undefeated and consistent within the defeasible reasoning chain.

5.3.1.21 Mixed Reasoning Chain

A reasoning chain is a mixed reasoning chain if it has a least one defeasible and one strict argument. Mathematically, a mixed reasoning chain can be represented as follows:

$$\forall r, s \in \lambda_{(r,j)}\{\exists r \textit{ that is a defeasible argument}, \exists s \textit{ that is a strict argument}\} \tag{5.22}$$

5.3.1.22 Dependent Reasoning Chains

A reasoning chain is dependent upon other reasoning chains if there is at least one common sub-argument. If the common argument is a strict argument, then a reasoning chain is known as a strictly dependent reasoning chain; if it is defeasible argument, then it is weakly dependent and medium dependent if it contains more than one common argument and those common arguments include both strict and defeasible arguments. Mathematically, this is represented as follows:

$$\textit{if } (\lambda_{(J,j)} \cap \lambda_{(H,h)}) \neq \varnothing \textit{ then } \lambda_{(J,j)} \textit{ and } \lambda_{(H,h)} \textit{ are dependent reasoning chains}. \tag{5.23}$$

5.3.2 Working of the Proposed Framework for Web@IDSS

In this section, the working of the proposed framework for Web@IDSS that can perform argumentative reasoning over incomplete and/or contradictory information which exists within an enterprise and/or in other enterprises is discussed and considered in decision making. As mentioned in Sect. 4.4.1, the proposed framework uses the DeLP language to represent incomplete and/or contradictory information in a declarative format, and uses a hybrid reasoning engine to reason over it. Figure 5.6 presents a flowchart diagram of the working of the proposed framework. The sequence of steps in the proposed framework are as follows:

1. Information representation in DeLP format
   The Web@IDSS, located at the Web-based DSS layer, takes into account the structured information located at the information layer. To achieve this objective, Web@IDSS exploits the functionality of the *information representation module* of the logic-based framework located at @IRRI layer. This module helps the Web@IDSS to translate the structured information in RuleML format into DeLP rules (also called as production rules) and saves them in the rule base i.e. $\mathcal{R}$. It also translates the structured information in OWL/RDF format to DeLP facts and saves them in the working memory i.e. $\mathcal{WM}$. Two translators have been developed to achieve this task as follows:
   - RuleML translator
     It translates the information specified in RuleML format to DeLP format. In most cases, the business rules of an enterprise are specified in RuleML format.
   - OWL/RDF translator
     It translates the information specified in OWL/RDF format to DeLP format. In most cases, the customer opinions, reviews/feedback about products and services offered by an enterprise are specified in OWL/RDF format.
2. Argumentative production system to perform hybrid reasoning
   Once the knowledge base (i.e. rule base containing DeLP rules and working memory containing DeLP facts) is formed, Web@IDSS exploits the functionality of the *argumentative reasoning module* of the logic-based framework to reason over the information present in the knowledge base. This process involves the following steps:
   - Arguments construction using data-driven reasoning.
     As mentioned previously, the information in the knowledge base may be potentially incomplete and/or contradictory (representing different viewpoints against a single issue) and current Web-based DSS are not able to perform reasoning over it. As a result, they can't assist the decision maker in the decision-making process. So, there is need to perform reasoning over such information and transform it into a format that is easily understood by the decision maker and can assist him in the decision-making process.

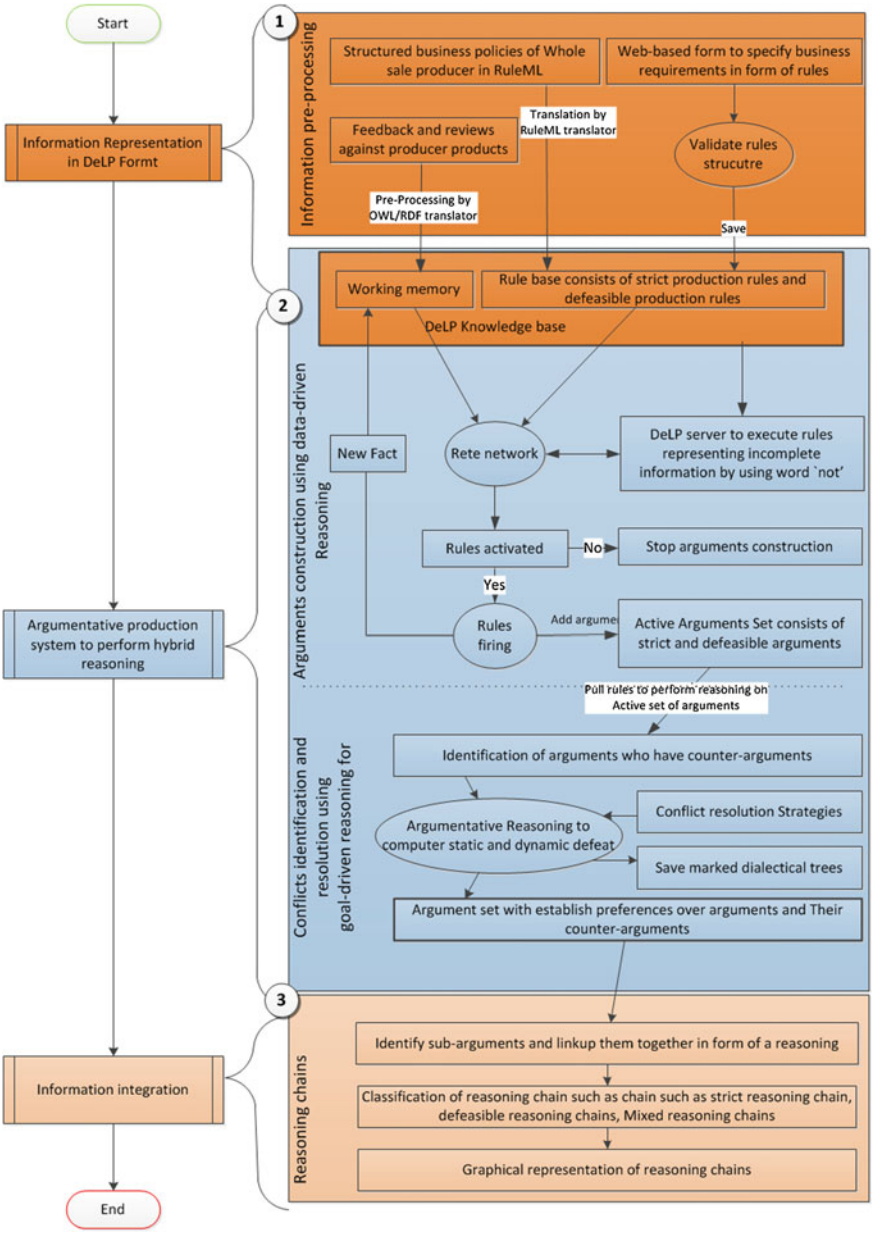

Fig. 5.6 Flowchart illustrating steps performed by Web@IDSS for information representation, reasoning and integration

In the proposed framework, this objective is achieved by transforming the incomplete and/or contradictory information in the knowledge base into a set of arguments. These arguments represent the different viewpoints in the underlying information in a declarative format. In the proposed framework, the construction of arguments involves two steps:

– Compilation of DeLP rules in the form of a Rete network.
– Perform data-driven reasoning by introducing certain DeLP facts from the working memory to the Rete network.

Data-driven reasoning results in the construction of a set of arguments supporting different conclusions. Two types of arguments which are constructed during this phase are strict arguments and defeasible arguments.

- Conflict identification and resolution using goal-driven reasoning

  Once the construction of arguments is complete, arguments which have counter-arguments are identified in order to resolve the conflicts between them and determine which one of them is defeated. To achieve this objective, two types of defeats are defined:

  – static defeat: a strict argument defeats a defeasible argument;
  – dynamic defeat: when there are two defeasible arguments in conflict with each other, then goal-driven reasoning is performed that uses a 'generalize specificity' conflict resolution strategy to resolve the conflict between them. During this process, a dialectical tree is constructed against the defeasible arguments that are in conflict and afterwards, each dialectical tree is marked as defeated or undefeated. These marked dialectical trees are used by the argumentative production system to resolve the conflict. The marked dialectical tree is then saved for future use, such as to provide an explanation for conflict resolution.

3. Information integration

Once the conflicts have been resolved, Web@IDSS exploits the functionality of the *information and knowledge integration module* of the logic-based framework to integrate the information obtained from hybrid reasoning and display it to the decision maker. This process involves the following steps:

- Construction of reasoning chains

  Once the hybrid reasoning is finished and conflicts have been resolved between arguments, the arguments need to be linked in the form of a chain. This module provides the functionality to link these arguments (supporting a conclusion) in the form of a chain know as a reasoning chain. During the construction of reasoning chains, different arguments supporting different conclusions result in the construction of different reasoning chains.
- Categorization of reasoning chains

  After the construction of reasoning chains, the next step performed by this module is to classify them on the basis of arguments upon which they are

built. The four categories defined in the proposed framework to categorise the reasoning chains are as follows:

- strict reasoning chains: composed of strict arguments only;
- defeasible reasoning chains: composed of defeasible arguments only;
- mixed reasoning chains: composed of at least one strict argument and one defeasible argument;
- dependable reasoning chains: composed of at least one argument that is shared with another reasoning chain.

- Graphical representation of reasoning chains

  The last functionality performed by this module is the graphical representation of reasoning chains for the decision maker. This will assist the decision maker in understanding the conclusion of the reasoning process and how that conclusion has been reached. Additionally, such representation of a reasoning process will help the decision maker to easily communicate the results to non-technical people such as the CEO of an enterprise.

In the next sections, the working of each of these steps defined in the proposed framework for Web@IDSS will be discussed in detail.

5.4 Information Representation in DeLP Format

As discussed in Sect. 4.4.1, the DeLP language is used in the proposed framework to represent incomplete and/contradictory information in Web@IDSS. The structured information is in RuleML and OWL/RDF format. There is need for a translation mechanism to translate that information into DeLP format and use it in the decision-making process. The proposed framework addresses this drawback with the help of the *information representation module* of the logic-based framework located at the @IRRI layer. There are two ways to represent information in Web@IDSS as shown in the Fig. 5.7. They are:

1. Information pre-processing

   During information pre-processing, the structured information is parsed and translated to DeLP format by using either the RuleML or OWL/RDF translator and is saved in the knowledge base.
2. Web-based form to specify DeLP rules and facts

   There may be some situations where there is no existing information available to be translated into DeLP format. In such cases, the decision maker has to specify the production rules by himself, depending on the objectives he wants to achieve. In the proposed framework, this objective is achieved by using a Web-based form to specify the DeLP rules and facts from scratch and saving them in the knowledge base.

In the next-subsections, each of these will be discussed in detail.

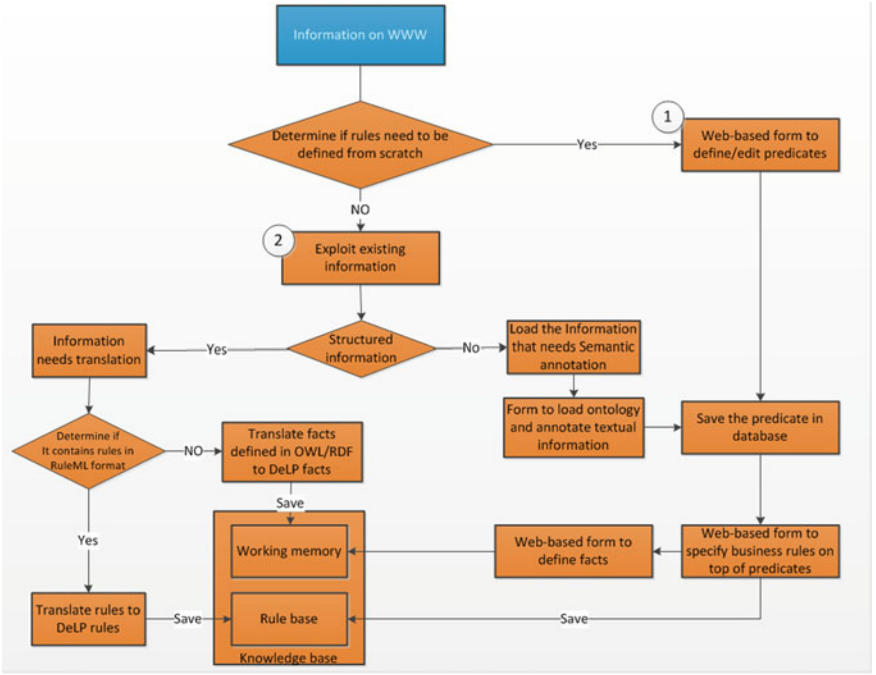

Fig. 5.7 Flowchart illustrating steps for information representation in Web@IDSS

5.4.1 Information Pre-processing

Information processing is carried out with the help of the following two translators:

(a) RuleML translator

RuleML supports different rule types via the 'implies' element and allows them to be named using the 'oid' element. RuleML syntax has been extended to express defeasible rules and superiority relations (Bassiliades et al. 2004; Pham et al. 2008). A '@ruletype' attribute has been added to the 'implies' element, allowing it to take one of three values: strict rule, defeasible rule or defeater. For the translation of rules to DeLP format, the RuleML translator performs the following steps:

- It loads the RuleML file (XML format) and starts its paring from the root element.
- It iterates through all the rules specified in the RulML file and by looking at '@ruletype' tag, it classifies them as either strict production rules or defeasible production rules. If '@ruletype' is absent, then it considers that production rule as strict.
- Then, it takes up each parsed production rule and starts building production rules in DeLP format. Information with the 'Rel' tag is captured as a predicate

businesspolicy.txt - Notepad

File Edit Format View Help

```
<?xml version="1.0" encoding="UTF-8"?>
<RuleML xmlns="http://www.ruleml.org/0.91/xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.ruleml.org/0.91/xsd http://debii.curtin.edu.au/~naeem/delp-valid-xsd.xsd">
    <Assert>
        <Implies ruletype="defeasiblerule">| <oid><Ind>d9</Ind></oid>
            <head>
                <Atom><Rel>purchase</Rel><Var>X</Var><Var>Y</Var></Atom>
            </head>
             <body>
                  <Atom><Rel>shopper</Rel><Var>X</Var></Atom>
                  <Atom><Rel>product</Rel><Var>Y</Var></Atom>
                 <Atom><Rel>havefeedback</Rel><Var>F</Var><Var>Y</Var></Atom>
                  <Atom><Rel>reviewRate</Rel><Var>G</Var><Var>F</Var></Atom>
            </body>
        </Implies>
        <Implies ruletype="defeasiblerule">
            <oid><Ind>d1</Ind></oid>
            <head>
                <Atom><Rel>giveDiscount</Rel><Var>X</Var></Atom>
            </head>
            <body>
                <Atom><Rel>shopper</Rel><Var>X</Var></Atom>
                <Atom><Rel>purchase</Rel><Var>X</Var><Var>Y</Var></Atom>
            </body>
        </Implies>
        <Implies ruletype="defeasiblerule">
            <oid><Ind>d2</Ind></oid>
            <head>
                <Neg>
                    <Atom><Rel>giveDiscount</Rel><Var>X</Var></Atom>
                </Neg>
            </head>
            <body>
                    <Atom><Rel>shopper</Rel><Var>X</Var></Atom>
                    <Atom><Rel>purchase</Rel><Var>X</Var><Var>Y</Var></Atom>
                <Atom>
                <Neg>
                    <Rel>advancePayment</Rel><Var>X</Var><Var>Y</Var>
                </Neg>
                </Atom>
            </body>
        </Implies>
        <Implies ruletype="defeasiblerule">
            <oid><Ind>d3</Ind></oid>
            <head>
                <Atom><Rel>giveDiscount</Rel><Var>X</Var></Atom>
            </head>
            <body>
                    <Atom><Rel>purchase</Rel><Var>X</Var><Var>Y</Var></Atom>
                    <Atom><Rel>shopper</Rel><Var>X</Var></Atom>
                    <Atom><Rel>bulkOrder</Rel><Var>X</Var><Var>Y</Var></Atom>
            </body>
        </Implies>
        <Implies ruletype="defeasiblerule">
            <oid><Ind>d5</Ind></oid>
           <head>
                <Neg>
                    <Atom><Rel>gstFree</Rel><Var>Y</Var></Atom>
                </Neg>
            </head>
            <body>
                <Atom><Rel>product</Rel><Var>Y</Var></Atom>
                <Atom><Rel>eShop</Rel><Var>Z</Var></Atom>
                    <Atom>
                  <Neg>
                    <Rel>packaging</Rel><Var>Y</Var><Var>Z</Var>
                  </Neg>
                </Atom>
            </body>
        </Implies>
        <Implies ruletype="strictrule">
            <oid><Ind>s1</Ind></oid>
            <head>
           <Atom><Rel>normalDiscount</Rel><Var>X</Var></Atom>
```

Fig. 5.8 Business policy of the supplier specified in RuleML format

in the body of the production rule and information with the 'Var' tag is captured as the subject and object variables in a predicate. If a parsed production rule head contains the 'Neg' tag, this is captured with the symbol '$\sim$' in the rule's conclusion. It this tag is found in the body of the parsed rule, it is captured with the symbol 'not' in the rule's body.

- After translation of each parsed rule to DeLP format, the production rules are then saved in the knowledge base.

The RuleML translator also saves other information about the RuleML file such as file URL, the number of rules translated, the owner/creator of rules etc., in a database for their profiling.

Table 5.1 Description of the supplier's production rules translated by the RuleML translator

Rule label	Description
d2	If a shopper does not pay in advance for the product, he may not receive a discount
d3	If a shopper purchases a product in bulk, he may be given a discount
d4	If a product needs packaging in the shop, then GST may apply
d5	If a product does not need packaging, then GST may not apply
s2	If GST applies and the shopper has been given a discount, then he must receive an ordinary discount
s1	If there is no information about GST, then the shopper must be given an ordinary discount
d7	If the shopper is given a normal discount, then he may be eligible to receive a platinum discount
d8	If the shopper plans to pay in installments (i.e. slow to pay) then he may be not be given a platinum discount

To explain with an example, consider the case study mentioned in Sect. 5.2 where Mr. David has to download and consider the business policies of a supplier in the decision-making process. Figure 5.8 shows the representation of the business policy of a supplier in RuleML.

The Web@IDSS downloads and translates the supplier's policies specified in the RuleML to DeLP rules and saves them in the rule base. Illustration 5.1 represents the set of DeLP rules extracted from the supplier's policy RuleML file.

$$\mathcal{R} = \left\{ \begin{array}{l} [d2]shopper(X), product(Y), not\ advancePyament(X,Y) \dashrightarrow\!\!\sim giveDiscount(X) \\ [d3]shopper(X), purhcase(X,Y), bulkOrder(X,Y) \dashrightarrow giveDiscount(X) \\ [d4]eShop(Z), packaging(Y,Z) \dashrightarrow gstFree(Y) \\ [d5]eShop(Z), not\ packaging(Y,Z) \dashrightarrow\!\!\sim gstFree(Y) \\ [s2]gstFree(Y), giveDiscount(X) \rightarrow ordinaryDiscount(X) \\ [s1]not\ gstFree(Y), giveDiscount(X) \rightarrow normalDiscount(X) \\ [d7]shopper(X), normalDiscount(X) \dashrightarrow platinumDiscount(X) \\ [d8]shopper(X), normalDiscount(X), plansSlowToPay(X) \dashrightarrow\!\!\sim platinumDiscount(X) \end{array} \right\} Illustration \quad (5.1)$$

$\mathcal{R}$, in Illustration 5.1, represents the rule base comprising strict and defeasible production rules. The rules with labels 's1' and 's2' are strict production rules, whereas the rest are defeasible production rules. Table 5.1 provides a description of each production rule in a natural language format.

(b) OWL/RDF translator

Translation of OWL/RDF information into DeLP facts using the following steps:

(a) OWL/RDF information is transformed with the help of the SWI-Prolog RDF Parser (Wielemaker 2011) into an intermediate triple format i.e. *rdf* (*Subject, Predicate, Object*).

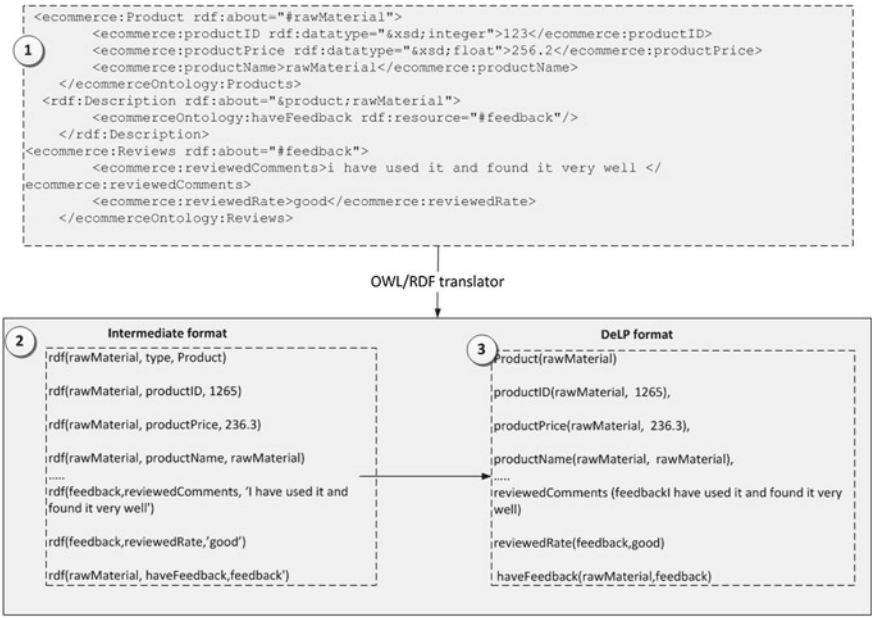

Fig. 5.9 Pictorial representation of the process for translation of information in OWL/RDF format to DeLP facts

(b) The intermediate triple format is further processed to transform the RDF statements into *Predicate*(*Subject, Object*) format.
(c) The facts in *Predicate*(*Subject, Object*) format are then saved in the knowledge base. The type attribute is further translated using the following formula: type(X, C)→C(X)

To explain with an example, consider the case study mentioned in Sect. 5.2, where Mr. David has to consider the customer's reviews/feedback about a supplier's product in his decision-making process. Figure 5.9 details the information representing the customer reviews/feedback about the supplier's product in RDF/XML format (represented as step 1). The OWL/RDF translator translates this information into an intermediate format (represented as step 2) and then into DeLP format (represented as step 3).
After translation of the customer's feedback, the OWL\RDF saves the DeLP facts in the working memory. Illustration 5.2 shows some of the DeLP facts that Mr. David will consider during the process of decision making.

$$\mathcal{WM} = \left\{ \begin{array}{l} eShop(BigW), product(rawMaterial) \\ havefeedback(rawMaterial, feedback), \\ reviwedRate(feedback, good) \end{array} \right\} \text{.....Illustration} \quad (5.2)$$

Table 5.2 provides a description of each DeLP fact shown in Illustration 5.2.

Table 5.2 Description of reviews/feedback by customer about supplier's production translated by OWL/RDF translator

DeLP fact	Description
eShop(BigW)	The eShop i.e. supplier providing the product, is BigW
product(rawMaterial)	The product is raw material
havefeedback(rawMaterial, feedback)	The raw material has some feedback
reviwedRate(feedback, good)	The feedback is good

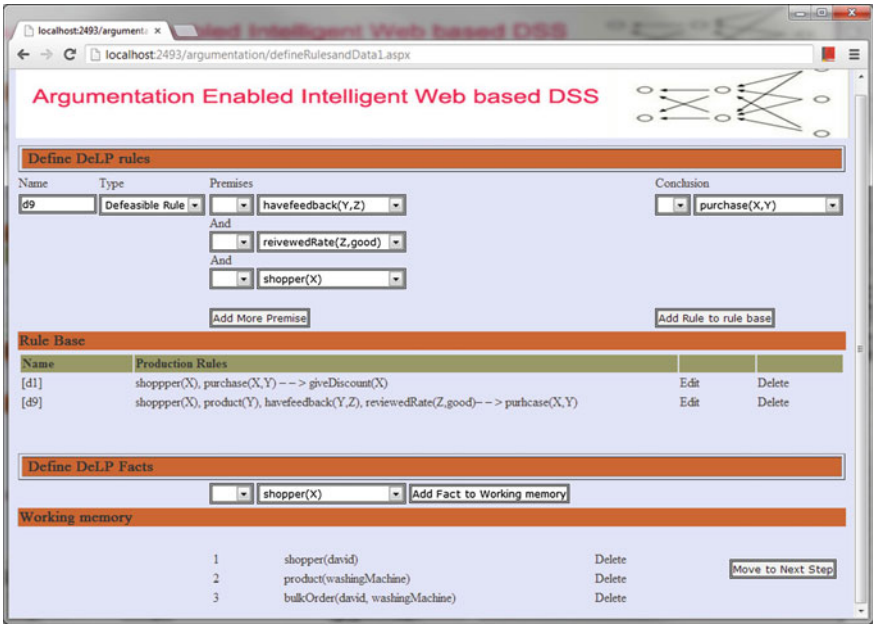

Fig. 5.10 Web-based form for the decision maker to specify DeLP rules and facts

5.4.2 Web-Based Form to Specify DeLP Rules and Facts

Another way by which information can be represented in DeLP format is by using the Web-based form of Web@IDSS as shown in Fig. 5.10. The Web-based form provides a GUI for the decision maker to define/edit DeLP rules and facts and saves them in the rule base and working memory, respectively. Using this form, Mr David can define his business requirements in the form of rules. For example, Mr David would like to purchase a product from a supplier who has good feedback from its customers. The following defeasible production rule captures his requirement in DeLP format:

- $[d9]shopper(X), product(Y), havefeedback(Y, Z), reviewedRate(Z, good) \dashrightarrow purchase(X, Y)$

Similarly, he wants to receive a discount on the purchase he may make. The following defeasible production rule captures his requirement in DeLP format:

- $[d1]shopper(X), purchase(X, Y) \dashrightarrow giveDiscount(X).$

Additionally, he also wants to specify that he may purchase the product in bulk. In DeLP language, such a parameter can be represented as a fact i.e. 'bulkOrder' using Web@IDSS form and saves it in the knowledge base.

5.5 Argumentative Production System to Perform Hybrid Reasoning

Once the required information for decision making has been captured in the knowledge base, then the next step is to perform reasoning over it. To address this objective, there is need for a hybrid reasoning methodology that can reason over the captured information and resolve any conflicts that may arise during the reasoning process. Web@IDSS achieves this objective with the help of an argumentative production system[1] that exploits the functionality of the *argumentative reasoning module* of the logic-based framework located at @IRRI layer. Figure 5.11 illustrates the steps performed by the argumentative production system for hybrid reasoning over the captured information. These steps are as follows:

1. Arguments construction using data-driven reasoning
   This step is further divided into the following two sub-steps:
   - The first step is the compilation of production rules in the rule base in the form of a Rete network. In the proposed framework, the Rete network has been extended to represent incomplete and/or contradictory information as Rete nodes in the Rete network. Additionally, the single production rule execution strategy of the Rete algorithm has been extended to execute all production rules that are activated during data-driven reasoning.
   - The second step is to perform data-driven reasoning over underlying information by passing the facts in the working memory through the Rete network. This results in the activation of production rules. The activation of production rules is followed by the firing of production rules. However, if the activated production rules' body represents some predicate starting with the symbol 'not', then before its firing, a query is sent to the DeLP server to compute its truthfulness by querying the knowledge base. If the query returns yes, then the production rule is fired, otherwise the activated production rules will be removed from the activated rule set. The firing of production rules results in the addition of new facts to the working memory and the instance of the production rule is stored as an argument in the 'argument set'. Data-driven

[1] In Sect. 3.3.1, some important definitions which will help the reader to understand the design and working of the argumentative production system are introduced.

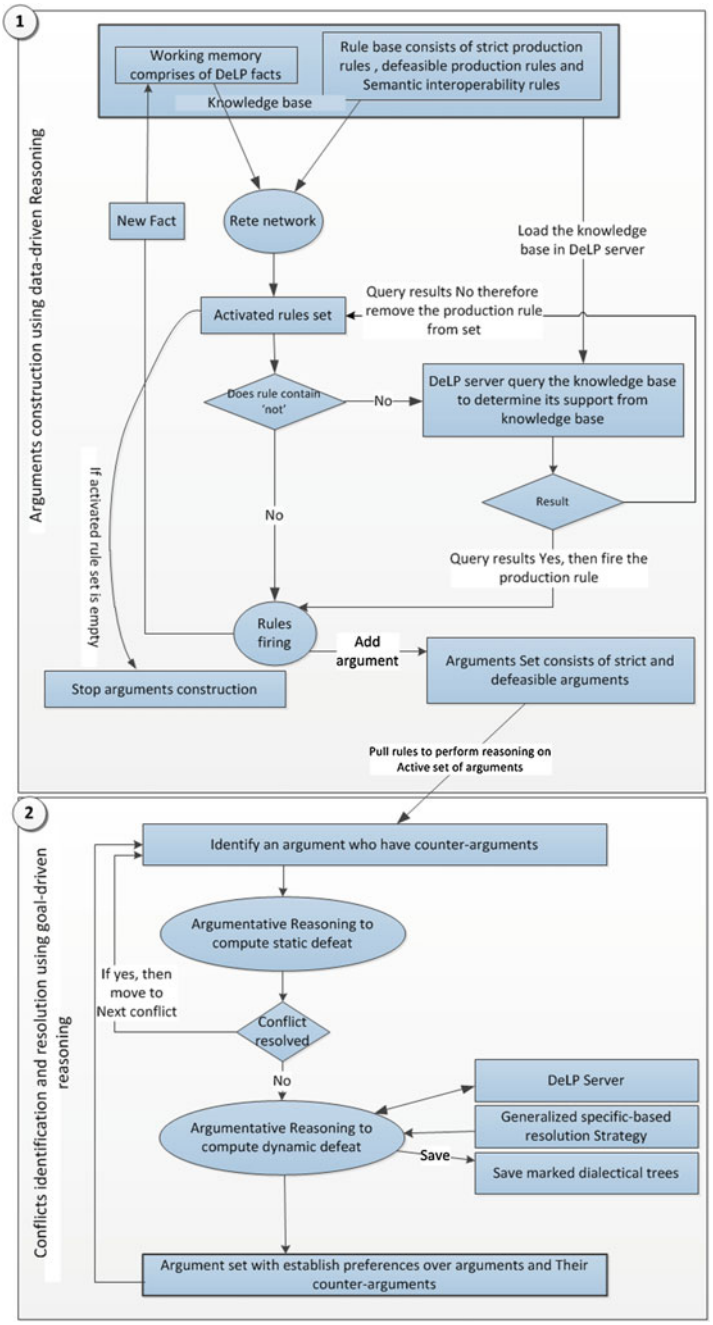

Fig. 5.11 Flowchart illustrating steps performed by Web@IDSS during hybrid reasoning

reasoning is a recursive process that continues until no further production rules are activated.

2. Conflicts identification and their resolution using goal-driven reasoning
   Conflicts identification and their resolution is a recursive process consisting of the following three steps:
   - Identification of an argument and its counter-argument.
   - If static defeat exits, then the conflict between argument and its counter-argument is resolved by establishing a preference between them and control flows back to the first step.
   - In the case where static defeat does not exist between an argument and its counter-argument, then dynamic defeat is computed by using the 'Generalize Specificity' conflict resolution strategy. The outcome of the dynamic defeat computation is the marked dialectical trees of an argument and its counter-argument that helps the argumentative production system to establish the priority between them. Once the priority has been established, the arguments are saved again in the argument set.

In next sub-sections, each of these steps will be discussed in detail.

5.5.1 Arguments Construction Using Data-Driven Reasoning

The construction of arguments from the knowledge base is a two-step process as follows:

1. Compilation of production rules in the form of a Rete network.
   The arguments construction process starts with the compilation of the production rules present in the rule base as a Rete Network. A general Rete Network (Cirstea et al. 2004) consists of a network of nodes, each of which represents one or more predicates that make up the body of the production rules as shown in Fig. 5.12. The three important nodes are as follows:
   - One-input nodes: These nodes are located at the first level of the Rete network and the facts from the working memory enter the Rete network through them. The different one-input nodes are as follows:
     - AssertCondition: The claim of a production rule is represented using this type of node.
     - RetractCondition: The claim can also be represented using this type of node if it may need to be removed later on from the working memory. In simple words, if contradictory information appears during the reasoning process, the general Rete network allows the removal of contradictory facts from the working memory in order to keep it consistent.
     - PositiveCondition: The predicates that make up the body of a production rule are represented using this type of node.

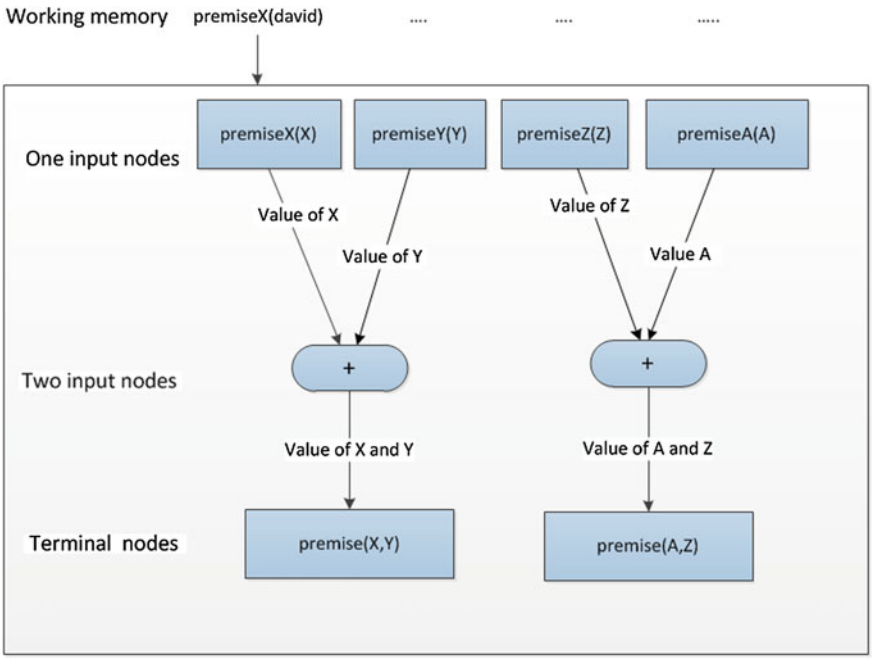

Fig. 5.12 Simplified representation of the compilation of production rules in a general Rete network

```
Variable x = new Variable("x");
Variable y = new Variable("y");
Variable z = new Variable("z");

Production prod1 = new Production();
prod1.LHS.Add(new PositiveCondition("", "shopper", x));
prod1.LHS.Add(new NegativeConditionNAF("", "advancePyament", y));
prod1.RHS.Add(new AssertCondition(y, "giveDiscount", x));
```

Fig. 5.13 Code snippet that shows a production rule with NegativeConditionNAF

- Two-input nodes: These are second level nodes in the Rete network and facts coming from the one-input node flow through to the two-input node and results in their activation.
- Terminal nodes: These are the last level nodes, each of which represent the claim of a production rule. When all the incoming two-input nodes to the terminal nodes are activated, it results in the activation of terminal nodes and the instantiated claim represented by a terminal node is added to the working memory.

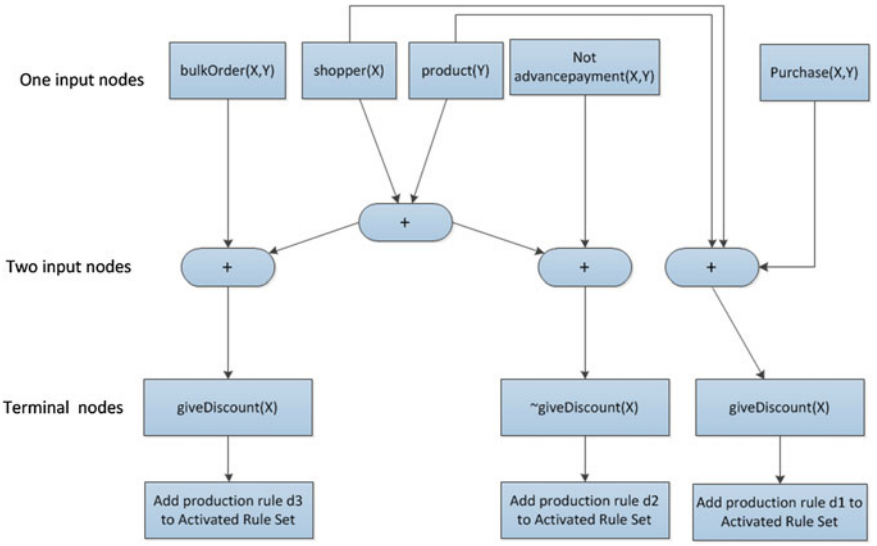

Fig. 5.14 Compilation of production rules in the form of a Rete network in Web@IDSS

In the proposed framework, the general Rete network has been extended to represent incomplete and/or contradictory information as Rete nodes in the network. The extensions made to one-input nodes are as follows:

- AssertCondition: The one-input nodes have been extend to represent contradictory information by introduction strong negation i.e. $\sim$, as an attribute in the AssertCondition class.
- NegativeConditionNAF: A new type of one-input node was introduced to indicate incomplete information represented by the symbol 'not'.

To explain the compilation of production rules in a Rete network, consider the rule base shown in Illustration 5.1 where defeasible production rules d2 and d3 are translated from the supplier's business policies and defeasible production rule d1 is specified by Mr. David using the Web-based form of Web@IDSS as shown in Fig. 5.10. Figure 5.14 shows the compilation of these three production rules in the form of a Rete network. The predicates that make up the body of the production rules such as bulkOrder(X, Y), shopper(X) etc are represented as one input node and the claim of the production rules d1, d2 and d3 are depicted as terminal nodes. The nodes in between the one-input node and the terminal nodes are represented as two-input nodes.

2. Perform data-driven reasoning over underlying information by passing the facts in the working memory through the Rete network

Once the production rules are compiled in the form of a Rete network, the next step is to perform data-driven reasoning by passing the facts in the working memory through the one-input nodes in the Rete network. This process, called

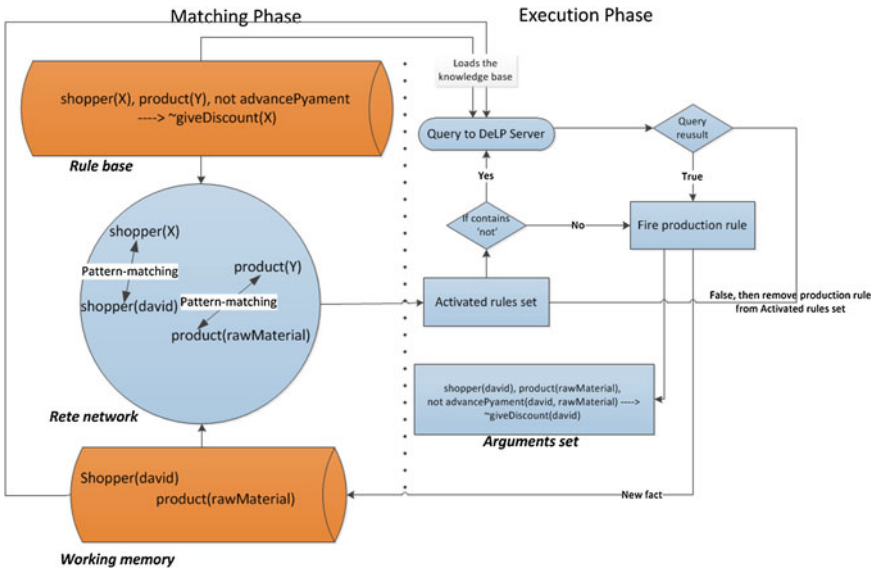

Fig. 5.15 Data-driven reasoning by passing the facts through the Rete network in Web@IDSS

data-driven reasoning, results in the activation of production rules called arguments (as defined in Sect. 5.3.1.9). Figure 5.15 illustrates the two important steps that are performed recursively during data-driven reasoning for the construction of arguments.

These steps are as follows:

- Matching phase: During this phase, pattern-matching is performed between the DeLP facts and the one input-node. If a pattern is matched, then the one-input node is activated and it forwards the value of the attributes to the two-input node. When two input-nodes receive the attribute value from all the incoming input nodes, it results in the activation of the respective terminal node.
- Execution phase: Once the terminal node is activated, the respective production rule is added to the Activated Rules set. The activation of production rules is followed by the firing of production rules. However, if the activated production rule's body represents a predicate starting with the symbol 'not', then before it is fired, a query is sent to the DeLP server to compute its truthfulness by querying the knowledge base. If it returns true, then the production rule is fired. Otherwise, it will be removed from the activated rules set. Firing the production rule will:
  - add a new fact to the working memory, and
  - add an instance of the rule to the argument Set.

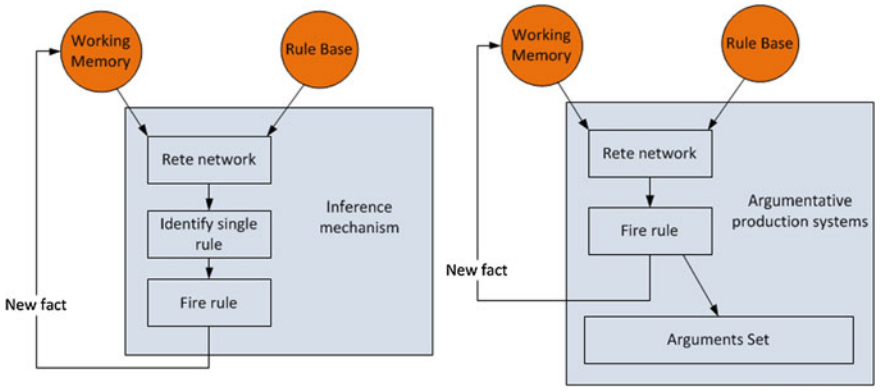

Fig. 5.16 Comparison of a standard Rete with a single rule execution strategy (*left*) with the extended Rete without the strategy (*right*)

It is important to note that in a general Rete network, data-driven reasoning works only on the production rules specified by an individual and the reasoning engine executes only one rule during a one match-execute cycle. If two rules are activated, the reasoning engine fires a production rule which has a higher preference specified by an individual at compilation time. In the proposed framework, the single rule execution strategy of the Rete network is removed as shown in Fig. 5.16. Therefore, if two contradictory production rules are activated, both will fire and instances of both production rules i.e. arguments, are added to the argument set.

Data-driven reasoning will stop when no more production rules are activated. A key issue to be noted here is that such new inferred facts may conflict with the existing knowledge base. The purpose is to retain contradictory information instead of eliminating it, in order to obtain better insight when deciding on business strategies.

To explain the working of data-driven reasoning for argument construction over underlying information as shown in Fig. 5.15, consider the following production rule in the rule base:

- $[i]shopper(X), product(Y), not \quad advancePayment(X, Y) \dashrightarrow \sim \quad give Discount(X)$

Further consider a working memory that contains facts such as $shopper(david)$ and $product(Y)$. During the matching phase, pattern-matching is performed in the Rete network between the DeLP facts and one-input nodes that comprise the body of the production rules. On successful matching, the production rule is activated and is added into the Activated Rules set. Once the matching phase is finished, then execution phase is started. During the execution phase, if the activated production rule represents some incomplete information such

as that represented by production rule i i.e. $not\ \ advancePayment(X, Y)$, then the predicate representing the incomplete information is passed as a query (i.e. $not\ \ advancePayment(X, Y)$) to the DeLP server. The DeLP server loads the knowledge base and executes the query on it. The current knowledge base does not have any information regarding advanced payment, therefore the DeLP query returns true and production rule 'i' is fired. The firing of the production rule adds the derived fact (e.g. derived predicate $\sim\ \ giveDiscount(david)$ as shown in Fig. 5.15), into the working memory and the instance of the production rule (e.g. an argument $shopper(david), product(rawMaterial), notadvancePyament(david, rawMaterial) \dashrightarrow giveDiscount(david)$ as shown in Fig. 5.15), is added to the arguments set.

Algorithm 5.1 demonstrates data-driven reasoning over underlying information by passing the facts in the working memory through the Rete network. It takes in the production rules specified in DeLP format and results in the construction of a set of arguments.

Algorithm 5.1: Argument construction using data-driven reasoning

Data: DeLP rules and DelP facts.
Result: Arguments.
initialization;
Construct Rete network alpha and beta nodes etc; initialize Rete network;
bool constructArgument ← true;
ActiveRules ActiveRuleSet;
ActiveArgumentSet ArgsSet;
repeat
  foreach *rule re* ∈ *KnowledgeBase* **do**
    if *match(re*, $\mathcal{WM}$*)=true* **then**
      ActiveRuleSet ← ActiveRuleSet ∪ *re*;
    else
      constructArgument ← false;
    end
  end
  foreach *re* ∈ *ActiveRuleSet* **do**
    $\mathcal{WM} \leftarrow \mathcal{WM} \cup claim(r)$; $ArgSet \leftarrow ArgSet + interpretationOf(re)$;
  end
until *constructArgument* ← *true*;

To explain argument construction using data-driven reasoning, consider an argumentative production system that captures information identified in the case study discussed in Sect. 5.2. This argumentative production system is named a $\mathbb{WEBIDSS}$. In light of the definition of an argumentative production system in Sect. 5.3.1.7, $\mathbb{WEBIDSS}$ can be defined as follows:

$$\mathbb{WEBIDSS} = (\mathcal{WM}, \mathcal{R}, Args) \tag{5.24}$$

where

$$\mathcal{WM} = \left\{ \begin{array}{l} shopper(david), eShop(BigW), product(rawMaterial) \\ havefeedback(rawMaterial, feedback), \\ reviwedRate(feedback, good), bulkOrder(david, rawMaterial) \end{array} \right\} ..Illustration \quad (5.3)$$

$$\mathcal{R} = \left\{ \begin{array}{l} [s2]gstFree(Y), giveDiscount(X) \rightarrow ordinaryDiscount(X) \\ [s1]notgstFree(Y), giveDiscount(X) \rightarrow normalDiscount(X) \\ [d1]shopper(X), product(Y), purchase(X, Y) \dashrightarrow giveDiscount(X) \\ [d2]shopper(X), not\ advancePayment(X, Y) \dashrightarrow\sim giveDiscount(X) \\ [d3]shopper(X), purchase(X, Y), bulkOrder(X, Y) \dashrightarrow giveDiscount(X) \\ [d4]eShop(Z), packaging(Y, Z) \dashrightarrow gstFree(Y) \\ [d5]eShop(Z), not\ packaging(Y, Z) \dashrightarrow\sim gstFree(Y) \\ [d7]shopper(X), normalDiscount(X) \dashrightarrow platinumDiscount(X) \\ [d8]shopper(X), normalDiscount(X), slowToPay(X) \\ \dashrightarrow\sim platinumDiscount(X) \\ [d9]shopper(X),\ product(Y), havefeedback(Y, Z) \\ reviwedRate(Z, good) \dashrightarrow purchase(X, Y) \end{array} \right\} Illustration \quad (5.4)$$

$$Args = ...Illustration \quad (5.5)$$

The $\mathcal{WM}$ represents the working memory of $\mathbb{WEBIDSS}$ and contains DeLP facts as shown in Illustration 5.3. $\mathcal{R}$ represents the rule base of $\mathbb{WEBIDSS}$ and contains production rules in DeLP format as shown in Illustration 5.4. $Args$ represents the arguments set of $\mathbb{WEBIDSS}$ and contains no argument as shown in Illustration 5.5.

The $\mathbb{WEBIDSS}$ captures all the information i.e. feedback about the supplier's production and Mr. David's facts in $\mathcal{WM}$, the business policies of the service provider and Mr. David as production rules in $\mathcal{R}$, and an empty arguments set. Given the definition of consistency in Sect. 5.3.1.8, in $\mathbb{WEBIDSS}$ the set {s1, s2} is consistent, whereas set {d1, d2} is inconsistent. It is important to note that production rule 'd1' is defined by Mr. David and argument 'd2' is a production rule representing the supplier's policy.

After arguments construction using data-driven reasoning over information in $\mathbb{WEBIDSS}$, it results in the following:

$$\mathbb{WEBIDSS} = (\mathcal{WM}', \mathcal{R}, Args) \quad (5.25)$$

where $\mathcal{WM}'$ represents the new state of the working memory after the addition of the new inferred facts. The argumentative production system with the updated working memory and populated with the argumentation is as follows:

$$\mathcal{WM}\iota = \left\{ \begin{array}{l} shopper(david), eShop(BigW), product(rawMaterial) \\ havefeedback(rawMaterial, feedback), \\ reviwedRate(feedback, good), bulkOrder(david, rawMaterial) \\ purchase(david, rawMaterial), \sim gstFree(rawMaterial), \\ giveDiscount(david), \sim giveDiscount(david), \\ normalDiscount(david), latinumDiscount(david) \end{array} \right\} \text{.....Illustration} \quad (5.6)$$

$$\mathcal{R} = \left\{ \begin{array}{l} [s2]gstFree(Y), giveDiscount(X) \rightarrow ordinaryDiscount(X) \\ [s1]not\ gstFree(Y), giveDiscount(X) \rightarrow normalDiscount(X) \\ [d1]shopper(X), product(Y), purchase(X, Y) \dashrightarrow giveDiscount(X) \\ [d2]shopper(X), not\ advancePayment(X, Y) \dashrightarrow \sim giveDiscount(X) \\ [d3]shopper(X), purchase(X, Y), bulkOrder(X, Y) \dashrightarrow giveDiscount(X) \\ [d4]eShop(Z), packaging(Y, Z) \dashrightarrow gstFree(Y) \\ [d5]eShop(Z), not\ packaging(Y, Z) \dashrightarrow \sim gstFree(Y) \\ [d7]shopper(X), normalDiscount(X) \dashrightarrow platinumDiscount(X) \\ [d8]shopper(X), normalDiscount(X), slowToPay(X) \\ \dashrightarrow \sim platinumDiscount(X) \\ [d9]shopper(X), product(Y), havefeedback(Y, Z) \\ reviwedRate(Z, good) \dashrightarrow purchase(X, Y) \end{array} \right\} \text{.....Illustration} \quad (5.7)$$

$$Args = \left\{ \begin{array}{l} [d1]shopper(david), purchase(david, rawMaterial) \\ \dashrightarrow giveDiscount(david) \\ [d2]shopper(david), not\ advancePayment(david, rawMaterial) \\ \dashrightarrow \sim giveDiscount(david) \\ [d3]shopper(david), purchase(david, rawMaterial), \\ bulkOrder(david, rawMaterial) \dashrightarrow giveDiscount(david). \\ [d5]eShop(BigW), not\ packaging(BigW, rawMaterial) \dashrightarrow \\ \sim gstFree(rawMaterial) \\ [s1]notgstFree(rawMaterial), giveDiscount(david) \\ \rightarrow normalDiscount(david) \\ [d7]shopper(david), normalDiscount(david) \dashrightarrow \\ \quad platinumDiscount(david) \\ [d9]shopper(david), product(rawMaterial), \\ havefeedback(rawMaterial, feedback), \\ reviewRate(feedback, good) \dashrightarrow purchase(david, rawMaterial) \end{array} \right\} \text{.....Illustration} \quad (5.8)$$

Illustration 5.8 represents the set of arguments constructed during the arguments construction phase. From Illustration 5.8, it can be seen that argument 's1' is a strict argument and the rest of the arguments i.e. d1, d2, d3, d5, d7 and d9 are defeasible arguments. These arguments represent the viewpoints of the supplier against Mr. David's business requirements in relation to whether to give him a discount and if so, how much. To explain with help of example, consider the following three defeasible arguments from Illustration 5.8:

- $[d1]shopper(david), product(rawMaterial), purchase(david, raw\ Material) \dashrightarrow giveDiscount(david)$
- $[d2]shopper(david), product(rawMaterial), not\ advancePayment(david, rawMaterial) \dashrightarrow \sim\ giveDiscount(david)$

- $[d3]shopper(david), purchase(david, rawMaterial), bulkOrder(david, rawMaterial) \dashrightarrow giveDiscount(X)$

Argument 'd1' represents Mr. David's viewpoint on receiving a discount. It states that he purchases the raw material, and as a result, he expects a discount. However, argument 'd2' from the supplier states that Mr. David purchased the product but did not make a payment in advance, so he may not receive a discount. However, another arguments 'd3' from the supplier states that Mr. David purchased the product and placed a bulk order, so he may receive a discount.

5.5.2 *Conflicts Identification and Their Resolution Using Goal-Driven Reasoning*

Once the argument construction process is complete, the conflicts identification and resolution phase is initiated. This is a recursive process that consists of the identification of an argument and its counter-argument and resolving the conflict between them. During this process, the following two types of defeats are computed in order to resolve conflicts between arguments:

1. Static defeat

   As defined in Sect. 5.3.1.13, a static defeat exits between an argument and its counter-argument if a strict argument defeats its defeasible counter-argument. This results in the establishment of the priority of a strict argument over a defeasible counter-argument. To explain static defeat with help of an example, consider an argument s1 from Illustration 5.8 as follows:

   - $[s1]not\ gstFree(rawMaterial), giveDiscount(david) \dashrightarrow normalDiscount(david)$

   Further consider a defeasible argument that states that if Mr. David is eligible for a discount, then he may not be given a normal discount as he is a bad customer. The defeasible argument is represented as follows:

   - $[d11]badCustomer(david), giveDiscount(david) \dashrightarrow \sim normalDiscount(david)$

   Argument 's1' is now in conflict with argument 'd11', however, there exits static defeat between them because a strict argument always defeats its defeasible counter-argument. Therefore, Mr. David will receive a normal discount.
2. Dynamic defeat by using the Generalize Specificity conflict resolution strategy to resolve conflict

   If static defeat does not exist between an argument and its counter-argument, the argumentative production system resolves the conflict between them by computing dynamic defeat. To achieve this objective, the argumentative production system takes into account the DeLP built-in *General Specificity conflict resolution strategy* (Garcia and Simari 2004). In this strategy, an argument (itself or with

the help of other arguments) is considered specific over its counter-argument if it requires less information to reach the final result. To understand the working of this strategy, consider the following two arguments:

(a) $[p1r2]executiveManager(X) \dashrightarrow approveTravel(X)$ which states that if X is an executive manager, then he may approve travel.
(b) $[p1r1]executiveManager(X), universityOfficer(Y), authorise(X, Y) \dashrightarrow \sim approveTravel(X)$ which states that if X is an executive manager and he authorises Y who is a university officer to approve travel, then X may not approve travel.

Considering the General Specificity conflict resolution strategy, the $< p1r1, \sim approveTravel >$ argument is more specific than the < p1r2, approveTravel > if the following two conditions hold:

(1) for all strict rules i.e. H and facts F, $H \subseteq F$, if $approveTravel(jon)$ is derived defeasibly and no strict derivation of $approveTravel(jon)$ exists, then the defeasible derivation of $\sim approveTravel(jon)$ exists, and
(2) there exists $H^{'} \subseteq F$ such that on basis of $H^{'}$ drives defeasibly $\sim approve Travel(jon)$ and there is no strict derivation of it and it does not drive defeasibly $approveTravel(jon)$.

Using General Specificity, the conflict between each argument and its counter-argument is resolved and the priority is saved in the knowledge base. The next step is to build the dialectical trees (as defined in Sect. 5.3.1.14) in order to identify whether this priority is supported by the entire knowledge base of the argumentative production system (in simple words, is there any argument that may counter-argue the preferred argument). This objective is achieved as follows:

- The claim of a preferred argument is submitted to the DeLP server (along with its preference over the counter-argument) and in return, it gives the marked dialectical tree (as defined in Sect. 5.3.1.15) of an argument. Similarly, the same procedure is performed for its counter-argument.
- Once the argumentative production system has the marked dialectical trees for both the argument and its counter-argument, an argument is preferred over its counter-argument if the marked dialectical tree of an argument is undefeated and the marked dialectical tree of counter-argument is defeated. In the case where the marked dialectical tree of both the argument and its counter-argument are undefeated, those arguments are considered blocking arguments and the system needs human intervention to resolve the conflict between them.

Algorithm 5.2 provides the detailed working of dynamic defeat using the General Specificity conflict resolution strategy to resolve conflicts by taking into account Algorithm 5.3 i.e. building and marking dialectical trees. The marked dialectical tree for argument d1 with undefeated status is represented as $\Sigma_U(d1$, giveDiscount(david)).

Algorithm 5.2: Dynamic defeat using the General Specificity conflict resolution strategy

Data: Arguments set.
Result: Preference establishment
initialization;
foreach arg_i *in ArgSet* **do**
  if $arg_i \Diamond arg_{i+1}$ **then**
    $\Sigma_{status}(arg_i, h_i) \leftarrow$ *BuildDialecticalTree*(arg_i, h_i);
    $\Sigma_{status}(arg_{i+1}, h_{i+1}) \leftarrow$ *BuildDialecticalTree*(arg_{i+1}, h_{i+1});
    if $\Sigma_D(arg_i,h_i)$ *and* $\Sigma_U(arg_{i+1},h_{i+1})$ **then**
      $arg_{i+1} > arg_i$;
    end
    if $\Sigma_U(arg_i,h_i)$ *and* $\Sigma_D(arg_{i+1},h_{i+1})$ **then**
      $arg_i > arg_{i+1}$;
    end
    if $\Sigma_B(arg_i, h_i)$*and*$\Sigma_B(arg_{i+1}, h_{i+1})$ **then**
      $arg_i <> arg_{i+1}$
    end
  end
end

Algorithm 5.3: Building and marking of a dialectical tree

Data: $(\mathcal{A}, h)$
Result: $\Sigma_{status}(\mathcal{A}, h)$
Let C $\leftarrow$ get all counter-arguments of $(\mathcal{A}, h)$; **if** $C \neq \emptyset$ **then**
  while *there is no* $\Sigma_U(\mathcal{A}_i, h_i) \in C$ **do**
    for *every argument in C* **do**
      Let $(\mathcal{A}_i, h_i) \leftarrow$
      $minimal\ non-labelled\ element\ BuildDialecticalTree(\mathcal{A}_i, h_i)$ getting result as $\Sigma(\mathcal{A}_i, h_i)$;
      Put $\Sigma(\mathcal{A}_i, h_i)\xi\ (\mathcal{A}, h)$;
    end
    if *there exist some* $\Sigma_U(\mathcal{A}_i, h_i)$ **then**
      Set $\Sigma_D(\mathcal{A}, h)$
    else
      Set $\Sigma_U(\mathcal{A}, h)$
    end
  end
else
  $\Sigma(\mathcal{A}, h) = (\mathcal{A}, h)$;
  Set $\Sigma(\mathcal{A}, h) \leftarrow$ *defeated*
end

To explain conflict identification and resolution using goal-driven reasoning with the help of an example, consider Illustration 5.8, where argument 'd1' counter-

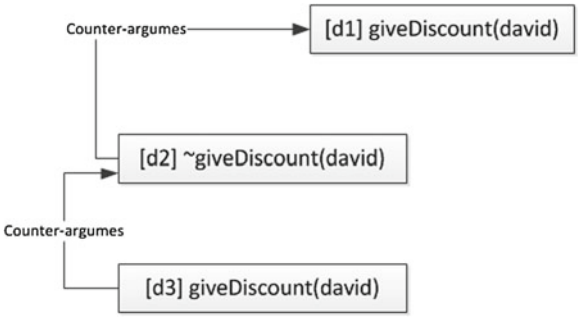

Fig. 5.17 Pictorial representation of arguments and their counter-arguments from Illustration 5.8

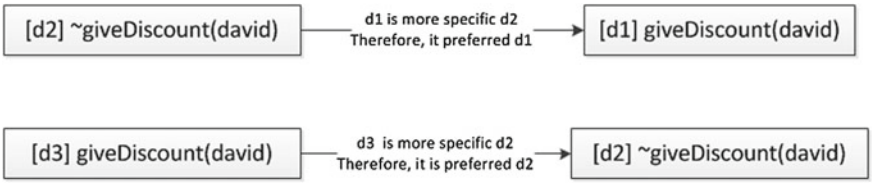

Fig. 5.18 Pictorial representation of preference between arguments using Generalize Specificity

argues argument 'd2', and argument 'd3' counter-argues argument 'd2', as shown in Fig. 5.17.

The WEBIDSS resolves the conflicts between the arguments and their counter-arguments using the General Specificity conflict resolution strategy. This task involves the following two steps:

(a) Identify conflict and establish the priority between an argument and its counter-arguments.

In this step, an argument e.g. d2, which is more specific than its counter-argument e.g. d1, defeats its counter-argument and results in priority establishment represented as follows: d2 > d1, as shown in Fig. 5.18. Similarly, argument 'd3' defeats argument 'd2' which results in its priority establishment as follows: d3 > d2.

(b) Building and marking of dialectical tress to obtain the priority status of an argument over its counter-argument by considering the entire knowledge base.

During this phase, the claims of argument d1 i.e. $giveDiscount(david)$ and the claims of its counter-argument d2 i.e. $\sim giveDiscount(david)$, along with the priority between them is sent to the DeLP server to perform goal-driven reasoning and it returns their marked dialectical trees. The construction of marked dialectical trees is defined in Sect. 5.3.1.15. Figure 5.19 shows a marked dialectical tree for argument 'd1' and 'd2'. In the figure, an argument is represented in the short form e.g. $[d1]giveDiscount(david)$ where [d1] is the label of the argument and $giveDiscount(david)$ is the claim of the argument.

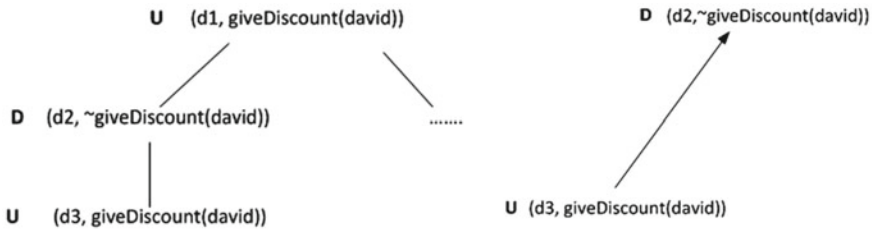

Fig. 5.19 Pictorial representation of undefeated marked dialectical tree for argument d1 (*left*), defeated marked dialectical tree for argument d2 (*right*)

It is evident from the figure that the marked dialectical tree for argument 'd1' is undefeated and the marked dialectical tree for argument 'd2' is marked as defeated in the tree. Therefore, argument d1 with the undefeated marked tree is preferred over argument 'd2' which has a defeated marked dialectical tree. It is important to note that in the undefeated marked dialectical tree for argument 'd1', argument 'd2' is marked as defeated. However, before the construction and marking of dialectical trees (as shown in Fig. 5.18), argument 'd2' was preferred over argument 'd1'. However, during this step, argument 'd2' is attacked by argument 'd3'. Argument 'd3' is more specific than argument 'd2', therefore, 'd3' is preferred over argument 'd2' (as shown in Fig. 5.18). The preference of argument 'd3' over 'd2' results in the revival of argument 'd1'. Therefore, in simple words, argument 'd3' supports argument 'd1' to withstand the attack of argument 'd2'. As a result, the marked dialectical tree for argument 'd1' is undefeated.

Once the marked dialectical trees are computed, argument 'd1' with the undefeated marked dialectical tree is preferred over its counter-argument 'd2' with the defeated marked dialectical tree. Such dialectical analysis of arguments helps decision makers such as Mr. David to understand that even though he may not have paid in advance (i.e. represented as argument $[d2]shopper(david)$, $product(rawMaterial)$, $not\ advancePayment(david, rawMaterial) \dashrightarrow\!\!\!\!\!\sim\ giveDiscount(david)$), by placing the bulk order (i.e. represented by argument $[d3]shopper(david)$, $purchase(david, rawMaterial)$, $bulkOrder(david, rawMaterial) \dashrightarrow giveDiscount(X)$), he may be offered a discount.

5.6 Information Integration

Once the argumentative production system has performed hybrid reasoning over the underlying information, the next step is to integrate the results of hybrid reasoning and display the results to the decision maker in a graphical format to assist him in the

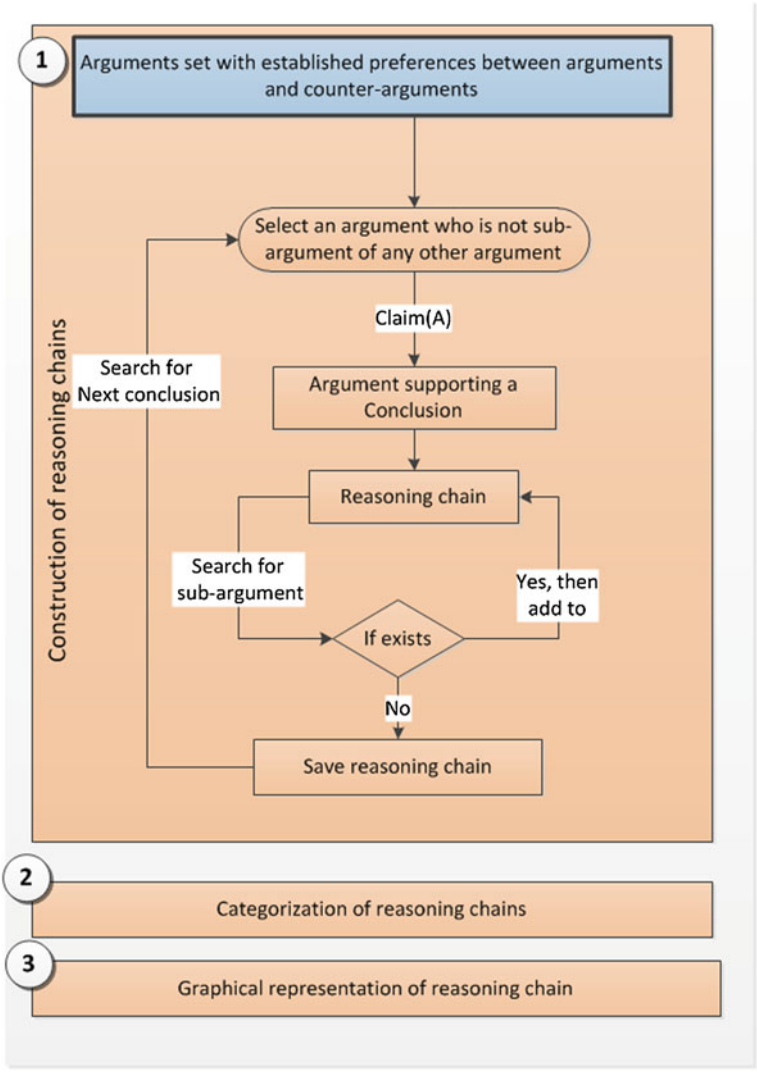

Fig. 5.20 Flowchart illustrating steps performed by Web@IDSS for information integration

decision-making process. To achieve this objective, Fig. 5.20 illustrates the following important steps in the proposed framework for Web@IDSS to integrate information:

1. Construction of reasoning chains

   During this step, the arguments are linked together in the form of a reasoning chain. Such reasoning chains help to link the initial information that triggers the reasoning process to reach the final conclusion. It also explains the important steps/information derived during reasoning to reach the final conclusion.

2. Categorization of reasoning chains
   Reasoning chains represent a decision supported by an argument or chain of arguments. Depending on the nature of argument/s supporting the decision, the argumentative production system can categorise a reasoning chain either as strict, defeasible, mixed or dependable. Such categorization of reasoning chains helps the decision maker to identify the strength/weakness of the information supporting the final decision.
3. Graphical representation of a reasoning chain
   During this step, the reasoning chain is depicted in a graphical format provide the decision maker with more easily comprehendible results. The graphical representation of results also helps him to easily communicate the results to higher authorities who may be non-technical people.

In the next subsections, each of these steps will be discussed in detail.

5.6.1 Construction of Reasoning Chains

The first step in information integration is the construction of reasoning chains. This process involves the following steps:

1. all sub-arguments with undefeated dialectical trees are linked together as a reasoning chain. This process will continue until all possible arguments are linked to form a reasoning chain;
2. the top argument i.e. conclusion of the reasoning chain is called the 'result' of the reasoning chain, and the chain of arguments supporting the top argument are called to support the conclusion;
3. ensure the reasoning chain is consistent (i.e., there is no contradiction in the result and support for the result).

Algorithm 5.4 provides the working of the construction of a reasoning chain by Web@IDSS.

Algorithm 5.4: Construction of a reasoning chain

Data: $(\mathcal{A}, h)$
Result: $\lambda_{(\mathcal{A},h)}$
Let S $\leftarrow$ get all sub-arguments of $(\mathcal{A}, h)$;
if $S \neq \emptyset$ **then**
  foreach $(\mathcal{A}_i, h_i) \in S$ **do**
    if *noCounterArgument*$(\mathcal{A}_i, h_i)$ *or* $\Sigma_U(\mathcal{A}_i, h_i)$ **then**
      BuildReasoningChain($(\mathcal{A}_i, h_i)$) ;
      Put $\lambda(\mathcal{A}_i, h_i)$ ξ $(\mathcal{A}, h)$;
    end
  end
else
  $\lambda_{(\mathcal{A},h)} = (\mathcal{A}, h)$;
end

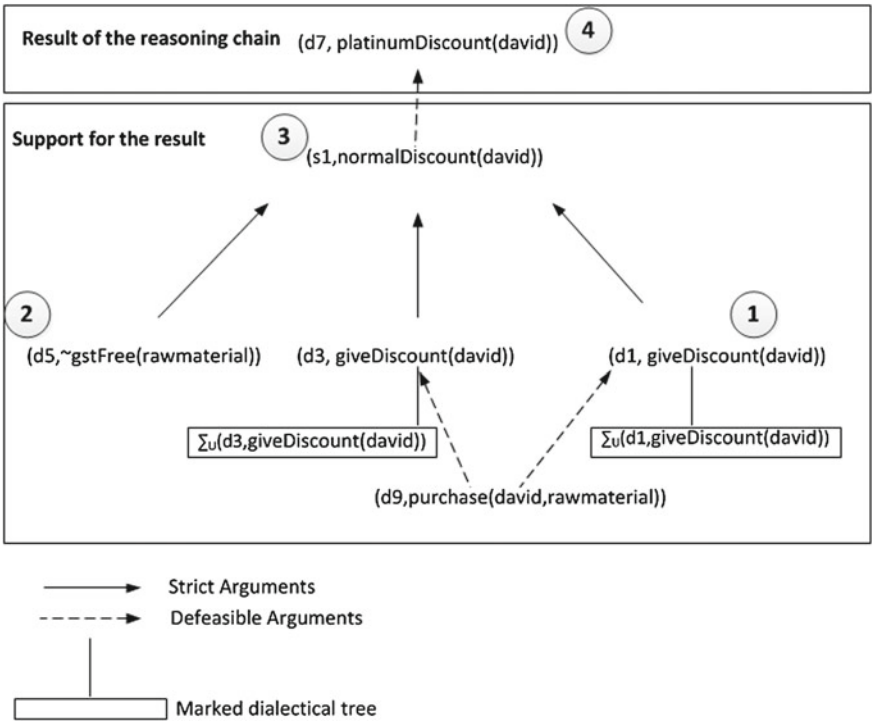

Fig. 5.21 Pictorial representation of mixed reasoning chain generated from arguments show in Illustration 5.8

To explain the construction of a reasoning chain, Figure 5.21 depicts a graphical representation of a reasoning chain constructed from the arguments shown in Illustration 5.8. The reasoning chain is divided into two parts, namely, the result of the reasoning chain i.e. platinum discount for Mr. David, and support for the result i.e. supporting information that backs the decision to give a platinum discount to Mr. David. By using such a graphical representation, Mr. David can identify that

- he may receive a discount by placing a bulk order (i.e. $[d3]shopper(david), purchase(david, rawMaterial), bulkOrder(david, rawMaterial) \dashrightarrow giveDiscount(X)$ and is depicted as 1 in Fig. 5.21);
- he can identify that the raw material he needs are not GST free, so as a result, he may be given a normal discount (i.e. $[s1]not\ gstFree(Y), giveDiscount(X) \rightarrow normalDiscount(X)$ and it is depicted as 1, 2 and 3 in Fig. 5.21);
- lastly, he can identify that if he receives a normal discount, there is a chance he may receive a platinum discount as well (i.e. $[d7]shopper(david), normalDiscount(david) \dashrightarrow platinumDiscount(david)$ as depicted as 3 and 4 in Fig. 5.21).

As mentioned in Sect. 5.3.1.18 where the formal definition of a reasoning chain was provided, it was pointed out that a reasoning chain should adhere to the following characteristics:

1. The reasoning chain should be consistent (i.e., there is no contradiction in the result and support for the result). Therefore, for example, $giveDiscount(david)$ and $\sim giveDiscount(david)$ will not belong to one reasoning chain, but each one of them can belong to different reasoning chains and those reasoning chains represent alternative paths or choices.
2. There is no defeated argument in a reasoning chain.
3. Two blocking arguments cannot be in the same reasoning chain.

5.6.2 Categorization of Reasoning Chains

A reasoning chain represents a set of small decisions linked to support the final conclusion. Depending on the nature of the arguments that are constructed during the reasoning process, the reasoning chains can be classified into the following different categories:

- Strict reasoning chain
  Such reasoning chains represent a decision process which cannot be challenged, even when new arguments are introduced into the argumentative production system.
- Defeasible reasoning chain
  Such reasoning chains represent decisions which can be challenged by the introduction of new arguments in argumentative production systems.
- Mixed reasoning chain
  Such reasoning chains have less weak points compared to defeasible reasoning chains which may be challenged by the introduction of new arguments in the argumentative production system which may result in a different conclusion.
- Dependent reasoning chain
  A reasoning chain is called dependent if it shares information with other reasoning chains. Such shared information points provide an alternative path in the decision-making process. If such information points are challenged by the introduction of new arguments in the argumentative production system, then the conclusions may change dramatically. Figure 5.22 shows two reasoning chains $\lambda_{(j3,h)}$ and $\lambda_{(s4,j)}$ sharing a common argument i.e. (s3, g).

To explain the categorization of reasoning chains with help of an example, consider a reasoning chain categorized as a mixed reasoning chain by WEBIDSS, and as depicted in Fig. 5.21. Such categorisation of a reasoning chain will help Mr David to identify the weak points (defeasible arguments) providing support to the overall conclusion. If new information arises later on, it may result in the defeat of those defeasible arguments, leading to different conclusions. For example, if he does not purchase in bulk, he will not only lose the opportunity to receive a discount, it will result in his failure to receive a normal discount and eventually a platinum discount.

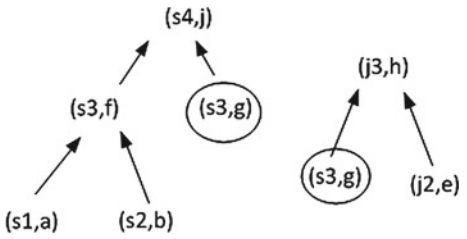

Fig. 5.22 Pictorial representation of dependent reasoning chains $\lambda_{(j3,h)}$ and $\lambda_{(s4,j)}$

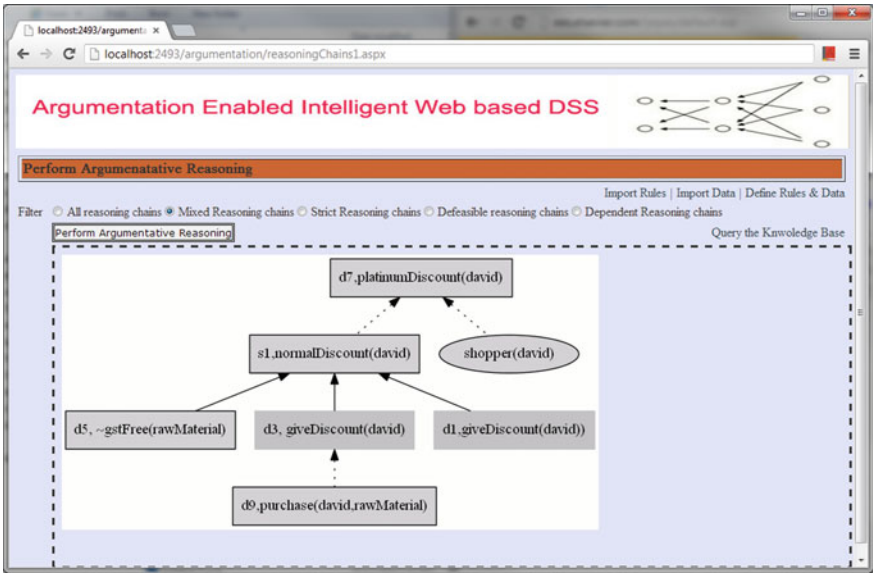

Fig. 5.23 Graphical representation of the reasoning chain generated by Web@IDSS

5.6.3 Graphical Representation of a Reasoning Chain

The last functionality performed by the information and knowledge integration module of the logic-based framework is the graphical representation of reasoning chains. To explain with help of an example, consider Fig. 5.23 which represents the graphical implementation of a reasoning chain depicted in Fig. 5.21. The important features of the graphical representation of a reasoning chain are as follows:

- The reasoning chain is represented as an inverted tree.
- An argument is represented in short form e.g. [$s1$]$normalDiscount(david)$ where [s1] is the label of the argument and $normalDiscount(david)$ is the claim of the argument.

- The *facts* are depicted as oval shapes, the *arguments* are depicted as rectangular shapes. The defeasible inference is depicted with a dotted arrow and strict inference as a straight arrow.

Such graphical representation helps Mr David to understand the whole reasoning process that results in a conclusion i.e. plantinumDiscount(david). He can easily identify the weak points (represented as defeasible inference) in a reasoning process where, if new information arises, this may result in the retraction of existing information, eventually leading to different results.

The graphical representation of the reasoning process takes into account the business policies of a supplier i.e. BigW, and the customer's feedback on its products in the decision-making process. In order to formulate a strategy for product B, Mr David has to perform the same activity with the rest of the suppliers in order to identify the supplier who may offer a maximum discount. As a result, he will obtain 'n' number of reasoning chains, each of which provides a different degree of discount under different conditions. By going through the graphical representations of the reasoning chains, Mr David can easily identify a supplier who may offer him a maximum discount considering his business requirements and the conditions with more strict rules. The graphical representation of the reasoning process will also help him to communicate his decision to the enterprise's CEO about why and how he reached the decision to select a particular supplier for raw material for the development of a new product.

5.7 Conclusion

In this chapter, the major shortcomings of the existing Web-based DSSs are addressed i.e. inability to represent and handle incomplete and/or contradictory information exists within an enterprise and/or in other enterprises. This is particularly important for those enterprises who take into consideration the information available on the Web for timely and accurate decision support. To overcome the limitations, the syntax and semantics of the DeLP language to represent, reason and integration information in Semantic Web applications is defined. A conceptual framework for 'Argumentation-enabled Web IDSS' is proposed and its workings are described in detail with the help of a case study.

In Chap. 3, I outlined certain research objectives that need to be addressed in order to support argumentation in Semantic Web applications. Some of the objectives have been addressed in this chapter and how they can be applied in Semantic Web applications for decision making when underlying information is inconsistent and incomplete are as follows:

- A methodology for incomplete and inconsistent information representation
  - A rule-based declarative language i.e. defeasible logic programming (DeLP) was selected for incomplete and/or inconsistent knowledge representation on the Semantic Web. DeLP allows information representation i.e. specifications

or preferences, that can be taken into account by the Web-based DSS and considered in the reasoning process to produce customized results for the decision maker.
  - A RuleML translator that translates the information defined in the RuleML to DeLP format was proposed. Such translation enables the exploitation of information which already exists on the Semantic Web specified in RuleML format.
  - An OWL/RDF translator that translates the information defined on the Semantic Web in the form of OWL and RDF into DeLP facts. The translated data is exploited by the rules during the reasoning process.

- A methodology for an argumentation driven-reasoning engine to reason over incomplete and inconsistent information
  - A hybrid reasoning engine to reason over information represented in DeLP format was proposed. The hybrid reasoning engine performs two types of reasoning: firstly, data-driven reasoning for argument construction and goal-driven reasoning for conflict identification between arguments and resolution.
  - A methodology to resolve conflicts among arguments by using the DeLP built-in Generalize Specificity conflict resolution strategy was proposed.

- Proposed a methodology for Information Integration
  - A mechanism to integrate the information being produced by different argumentation-driven hybrid reasoning engines was proposed and its graphical representation was provided to the decision maker to enhance their understanding of the reasoning process and results.

References

Antoniou G, Bikakis A (2007) Dr-prolog: a system for defeasible reasoning with rules and ontologies on the Semantic Web. IEEE Trans knowl Data Eng 19(2):233–245

Bassiliades N, Antoniou G, Vlahavas I (2004) Dr-device: a defeasible logic system for the semantic web. In: Principles and practice of semantic web reasoning, Lecture notes in computer science, vol 3208. Springer, Berlin, pp 134–148

Brodie M (2008a) Understanding our digital universe: unleashing natural forces. In: 2nd IEEE international conference on digital ecosystems and technologies, Phitsanulok, Thailand

Brodie ML (2008b) The end of the computing era: Hephaestus meets the olympians. In: Paige RF, Meyer B (eds) Lecture notes in business information processing, vol 11. Springer, Berlin, pp 0–1. doi:10.1109/DEST.2008.4635099

Cirstea H, Kirchner C, Moossen M, Moreau PE (2004) Production systems and rete algorithm formalisation. Contrat A04-R-546 cirstea04d. http://hal.inria.fr/inria-00099850. Last accessed 10/02/2012. http://hal.inria.fr/inria-00099850, rapport de contrat

Garcia AJ, Simari GR (2004) Defeasible logic programming: an argumentative approach. Theory Pract Log Program 4(1–2):95–138

Grosof B, Gandhe M, Finin T et al (2002) Sweetjess: translating damlruleml to Jess. In: Proceedings of the internationalworkshop on rule markup languages for business rules on the semantic web at 1st international semantic web conference, Sardinia, Italy, vol 60

Lee KC, Chung N (2005) A Web DSS approach to building an intelligent internet shopping mall by integrating virtual reality and avatar. Expert Syst Appl 28(2):333–346

Pham D, Governatori G, Raboczi S, Newman A, Thakur S (2008) On extending RuleML for modal defeasible logic. In: Bassiliades N, Governatori G, Paschke A (eds) Rule Representation, Interchange and Reasoning on the Web, Lecture notes in computer science, vol 5321. Springer, Berlin, pp 89–103

Power DJ (2002) Decision support systems: concepts and resources for managers. Greenwood Publishing Group, Westport

Power DJ, Sharda R (2009) Decision support systems. In: Nof SY (ed) Springer handbook of automation. Springer, Berlin, pp 1539–1548

Silverman BG, Bachann M, Al-Akharas K (2001) Implications of buyer decision theory for design of e-commerce websites. Int J Hum Comput Stud 55(5):815–844

Toni F (2007) E-business in argugrid. In: Veit D, Altmann J (eds) Grid economics and business models, Lecture notes in computer science, vol 4685. Springer, Berlin, pp 164–169

Vahidov R, Kersten GE (2004) Decision station: situating decision support systems. Decis Support Syst 38(2):283–303

Wielemaker J (2011) SWI-Prolog RDF parser. http://www.swi-prolog.org/pldoc/package/rdf2pl.html

Yao Y, Zhong N, Liu J, Ohsuga S (2001) Web Intelligence (WI) research challenges and trends in the new information age. In: Web intelligence: research and development, Lecture notes in computer science, vol 2198. Springer, Berlin, pp 1–17