The Car Repair Assistant

Joost Huizinga (1557017)
Tycho Bismeier (1615440)

February 22, 2011

Contents

1 Introduction

Cars are probably the most important vehicle of our century. Where most people would use cars only for transport some have turned their car in a hobby. For these hobbyists, performing minor upgrades and repairs on there cars is about as important as driving the car.

Unfortunately its is sometimes difficult to find the right information to perform repairs and upgrades. For this problem we developed the car repair assistant. The car repair assistant is a knowledge system that helps with the identification and repair of various problems a car might have.

Because car repair is a very wide problem area we have narrowed it down to only the electrical system. Most of our models nevertheless assume the complete car repair assistant. It are the domain specific models that only consider the electrical system of the car.

2 Context model

This section contains the tables OM-1, OM-2, OM-5, TM-1, TM-2, AM-1. Since our problem is based on a single person that repairs his car in its own time there is no organization. This makes organizational structures or flows quite impossible to create. Nevertheless, most parts of the forms that concern the organizational model are still relevant.

2.1 Organizational model

Organization Model	**Problems and Opportunities Worksheet OM-1**
Problems and opportunities	Hobbyists currently experience problems with the availability of documentation, reasoning about complex, partly unknown technical systems and gaps in their own knowledge.
Organizational context	The goal of the hobbyist is to repair his or her car. He wants to have fun and learn about cars. He might also save money because he does not have to go to a garage or he might be better than a garage in repairing his car because he has more specific knowledge about his car. The car needs to be in a good enough shape to pass the APK. Other users of the car require the car to be available. External factors like work and family restrict the time and resources available to repair in ways that are uncontrollable or unexpected.
Solutions	Possible solutions are: • An information retrieval system to find documentation about car repair and specific cars on the Internet. • A knowledge system holding knowledge about car repair, reasoning about that knowledge and using it to assist the hobbyist in repairing his car. • Educating the car hobbyist about car repair.

Organization Model	**Variant Aspects Worksheet OM-2**
Structure	The organizational structure is simple. The car hobbyist is working alone.
Process	There is the process of repairing a car.
People	The car hobbyist might occasionally have an assistant if he perform a task that requires it. Like checking whether the headlights are working.
Resources	Repair manuals, technical description of a specific car, general technical description of car. Tools.
Knowledge	The car hobbyist uses knowledge about how to repair a car, knowledge about how cars work in general and knowledge about how a specific car works.
Culture & power	Car repair happens in the social environment of his or her family.

Organization Model	**Checklist for Feasibility Decision Document: Worksheet OM-5**
BUSINESS FEASIBILITY	We expect that: car repairs are performed more quickly, car repairs with a higher difficulty can be performed and the hobbyist learns more while repairing his car. The car hobbyist spends less time on thinking about repairs and performing repairs that don't result in a repaired car. Depending on the hobbyist this might increase or decrease his fun in repairing the car. We don't know how much time might be saved, or how much more a hobbyist might learn. The hobbyist does need to have computer to run the knowledge system on. That computer also needs an interface that is usable with greasy hands. This compare favourably with the other solutions. The hobbyist would be prepared to spend money on a knowledge system, while the time needed for further eduction is not available, and a knowledge retrieval system would not have the same benefits. There is no need for organizational changes. The hobbyist still works on his own while repairing the car.
TECHNICAL FEASIBILITY	The system needs to perform diagnosis with causal reasoning. That can be done with a knowledge system. There is no need to reason with the knowledge about how to repair a car, this can be represented in text form to the user. The system needs to run on a PC sized computer. It should respond in seconds to response of the user. The system has minutes of reasoning time when the hobbyist is performing repairs. The system is successful if it can guide a car repair hobbyist in common repairs to the electrical system of one specific car. It also needs to be modular and modifiable enough, so that it's clear it can be extended to more special repairs, repairing more kinds of car and repairing non-electrical faults.
Project feasibility	The project is feasible. We have access to an expert on car repair and a car hobbyist.

2.2 Task and agent models

Task Model	**Task Analysis Worksheet TM-1**
Task	Car diagnoses task
Organization	This task is not part of an organization and is executed by a hobbyist.
Goal and value	To find the cause of an observed problem. Once the cause is know it might be possible to repair the problem or bring the car tho a garage so they can fix the problem.
Dependency and flow	*Input tasks*: None *Output tasks*: Car repair task
Objects handled	*Input objects*: An observed problem, type of car. *Output objects*: Cause(s) of the observed problem. *Internal objects*: General car knowledge, specific car knowledge
Timing and control	The task is performed only when a problem is observed, which is expected to be very infrequent. The task might take from about 15 min to a few hours. *preconditions* There must be an observed problem. *postconditions* One or more causes for the observed problem are reported or the system reports it could not find the problem. *constraints* The system needs access to the car knowledge of the specified car.
Agents	The hobbyist, the car diagnoses system
Knowledge and competence	The hobbyist needs to be able to preform certain tests, interpret the results correctly and give them to the car diagnoses system.
Resources	The task will consume the time of the person performing it. In addition, depending on the problem being diagnosed, some measuring equipment and/or spare parts might be needed.
Quality and performance	The quality measure will be that the identified cause is the real cause of the problem. The performance will be measured by the time it takes to diagnose the problem.

Task Model	**Task Analysis Worksheet TM-1**
Task	Car repair task
Organization	This task is not part of an organization and is executed by a hobbyist.
Goal and value	To repair the observed cause. If successful then the car functions properly again, saving repair costs
Dependency and flow	*Input tasks*: Car diagnoses task *Output tasks*: None
Objects handled	*Input objects*: A cause for a problem, type of car. *Output objects*: Confirmation if the task was successful *Internal objects*: General car knowledge, specific car knowledge, repair knowledge
Timing and control	The task is performed only when a problem is observed, which is expected to be very infrequent. The task might take from about 15 min to a few hours. *preconditions* A cause for a problem must be identified *postconditions* The problem is either fixed or it is known that the repair failed *constraints* The system needs access to the car knowledge of the specified car.
Agents	The hobbyist
Knowledge and competence	The hobbyist needs to be able to preform the necessary repairs
Resources	The task will consume the time of the person performing it. In addition, depending on the repair, some tools and/or parts might be needed.
Quality and performance	The quality measure will be that the problem is fixed by taking away the cause. The performance will be measured by the time it takes to perform the repair.

Task Model	**Knowledge Item Worksheet TM-2**
NAME	Component knowledge
EXPLANATION	Knowledge about what the individual components do. Knowing that the battery provides power and that if the battery is connected to lights then the lights will work is component knowledge. This knowledge does not include how the components are wired in a car.
POSSESSED BY	Car mechanics, some hobbyists, manufacturers
USED IN	Car diagnoses, car repair
DOMAIN	Cars

Nature of the knowledge		**Bottleneck / to be improved?**
Formal, rigorous	X	
Empirical, quantitative	X	
Heuristic, rules of thumb		
Highly specialized, domain-specific		
Experience-based		
Action-based		
Incomplete		
Uncertain, may be incorrect		
Quickly changing		
Hard to verify		
Tacit, hard to transfer		

Form of the knowledge		
Mind	X	
Paper	X	
Electronic	X	
Action skill		
Other		

Availability of knowledge		
Limitations in time		
Limitations in space		
Limitations in access		
Limitations in quality		
Limitations in form		

Task Model	**Knowledge Item Worksheet TM-2**
Name	General car knowledge
Explanation	Knowledge about the overall layout of cars in general. This knowledge will include the fact that car usually have a battery and that this battery is usually connected to the lights in one way or an other.
Possessed by	Car mechanics, most hobbyists
Used in	Car diagnoses
Domain	Cars

Nature of the knowledge		**Bottleneck / to be improved?**
Formal, rigorous		
Empirical, quantitative		
Heuristic, rules of thumb	X	
Highly specialized, domain-specific		
Experience-based	X	
Action-based		
Incomplete	X	X
Uncertain, may be incorrect	X	X
Quickly changing		
Hard to verify		
Tacit, hard to transfer		

Form of the knowledge		
Mind	X	
Paper	X	
Electronic	X	
Action skill		
Other		

Availability of knowledge		
Limitations in time		
Limitations in space		
Limitations in access		
Limitations in quality	X	X
Limitations in form		

Task Model	**Knowledge Item Worksheet TM-2**
Name	Specific car knowledge
Explanation	Knowledge about the exact layout of a specific car. This knowledge includes exactly what components are connected to what but it does not include what to components actually do.
Possessed by	Some car mechanics, some hobbyists, manufacturers
Used in	Car diagnoses
Domain	Cars

Nature of the knowledge		**Bottleneck / to be improved?**
Formal, rigorous	X	
Empirical, quantitative	X	
Heuristic, rules of thumb		
Highly specialized, domain-specific	X	
Experience-based		
Action-based		
Incomplete		
Uncertain, may be incorrect		
Quickly changing		
Hard to verify		
Tacit, hard to transfer		

Form of the knowledge		
Mind	X	
Paper	X	
Electronic	X	
Action skill		
Other		

Availability of knowledge		
Limitations in time		
Limitations in space		
Limitations in access	X	X
Limitations in quality		
Limitations in form		

Task Model	**Knowledge Item Worksheet TM-2**
NAME	Car Repair Knowledge
EXPLANATION	Knowledge about the best way to perform tasks that include finding, reaching and replacing parts. This knowledge also includes the way to handle tools like a wrench in the car repair domain.
POSSESSED BY	Car mechanics, some hobbyists
USED IN	Car diagnoses, car repair
DOMAIN	Cars

Nature of the knowledge		**Bottleneck / to be improved?**
Formal, rigorous	X	
Empirical, quantitative		
Heuristic, rules of thumb	X	X
Highly specialized, domain-specific		
Experience-based	X	X
Action-based	X	X
Incomplete		
Uncertain, may be incorrect		
Quickly changing		
Hard to verify		
Tacit, hard to transfer	X	X

Form of the knowledge		
Mind	X	
Paper	X	
Electronic	X	
Action skill	X	
Other		

Availability of knowledge		
Limitations in time		
Limitations in space		
Limitations in access		
Limitations in quality		
Limitations in form	X	X

Agent Model	**Agent Worksheet AM-1**
NAME	*Hobbyist*
ORGANIZATION	The agent is the human person physically executing the diagnoses. Usually the owner of the car.
INVOLVED IN	Car diagnoses, car repair
COMMUNICATES WITH	The car diagnoses system
KNOWLEDGE	Some very basic car knowledge
OTHER COMPETENCES	Able to perform simple tests and repairs
RESPONSIBILITIES AND CONSTRAINTS	The agent has the responsibility to make sure its car works correctly, without the risk of suddenly failing The agent might have the responsibility to make sure its car works at a certain time point because someone else wants to use it. The agent might be constrained by warranty voids when handling certain parts himself

3 Knowledge model

3.1 Domain knowledge

In our context analysis we identified four different kinds of knowledge shown in tables TM-2. This knowledge is all domain knowledge but they are represented in different ways in out model. The general car knowledge would be a car template, a concept that is part of our domain schema. Because we did not implement the ability for our system to reason with car templates we do not have an instantiation of such a template.

A simplified instantiation of specific car knowledge is shown in figures 3, 4 and 5. These figures show exactly which components exist and how they are connected in our example car. A part of this knowledge is also present in our knowledge base since the components since the mappings between component states and observables can be different for every car. There exist many different cars and every car would have its own scheme. Modeling them is beyond the scope of this project.

Component knowledge is contained in our knowledge base. These are the rules that state exactly what happens when two components are connected. Note that these rules do not specify specific components and they are thus independent of the actual car.

Car repair knowledge is not present in our model. This is because our system does not use car repair knowledge for reasoning. Adding car repair knowledge could enhance the system by enabling it to propose observables that are easy to observe first.

3.1.1 Domain schema

The domain schema shown in figure 1 shows that a car 'consists of' electrical parts and that these parts can be of several types. Connections are special because they are electrical components that connect other components. The rule types shown in figure 2 define the rules that exist between these concepts. The direct causal dependendies, 'causes to wire' and 'causes from wire', can only exists between between a wire and a component. There are no direct causal dependencies between components. A causal dependency between two components always runs trough a wire. The manifestion rules tells that if a component is in a certain states a particular observation can be made.

3.1.2 Knowledge base

Below is a general overview of our knowledge base. Because all rules together create a single causal model all rules are contained in a single knowledge base. Not all rules are shown. Instead we show one instance of every kind of rule and explain how the other rules look. This is done because the complete knowledge base is very large.

The CAUSES-TO-WIRE rules state that if a device is generating or passing through power and this device is in a state that makes it unable to perform this function then all wires leading from it will not be powered. A separate rule exists for the combinations of each device type with every possible state that would make it unable to power the wire.

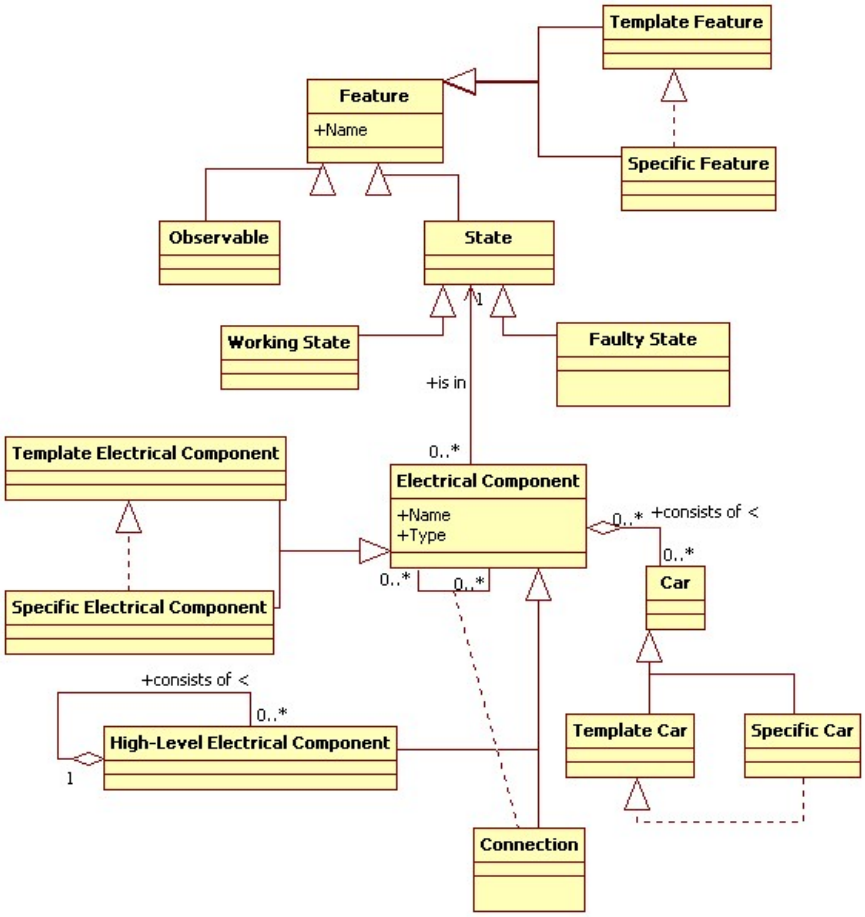

Figure 1: Domain schema for the car diagnoses task

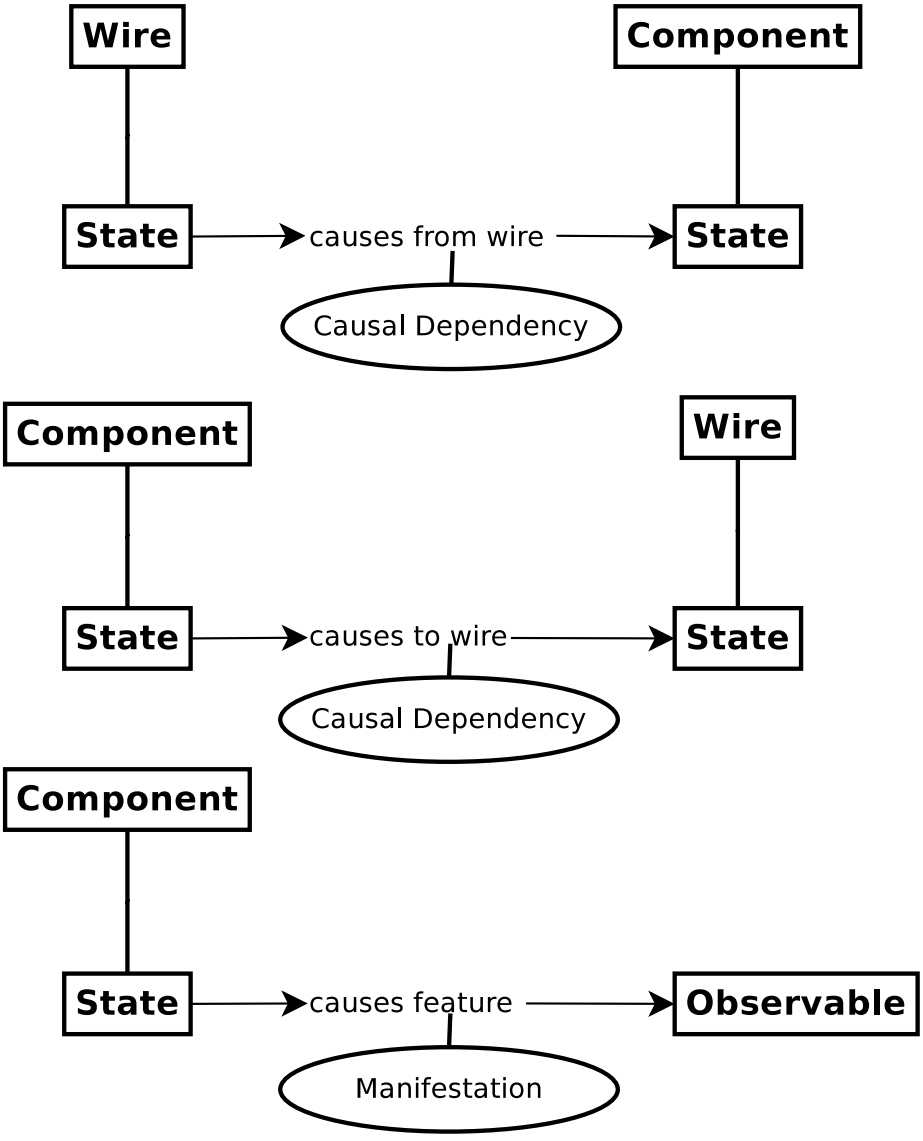

Figure 2: Rule types

```
KNOWLEDGE-BASE causal-model;
   USES:
      Component-knowledge
      Car-specific-knowledge
   EXPRESSIONS:
      A.state = empty
      A.type = battery
      B.type = connection
      B.comp1 = A
      CAUSES-TO-WIRE
      B.state = no-power
```

The CAUSES-FROM-WIRE rules state that if all wires connected to a device are in a state that makes them unable to pass on power then this device will not be powered. A separate rule exists for the combination of every wire state that makes the wire unable to pass on power and every device type.

```
      A.state = empty
      A.type = battery
      B.type = connection
      B.comp1 = A
      CAUSES-FROM-WIRE
      B.state = no-power
```

The CAUSES-FEATURE rules map states of actual devices to observables. A separate rule exists for the combination of every device with all of its possible states that cause an observable. Note that these rules are not general and are dependend on the car. Thus these rules represent specific car knowledge.

```
      A.name = head-light-left
      A.state = no-power
      CAUSES-FEATURE
      observable = head-light-left-no-light
END KNOWLEDGE-BASE causal-model;
```

3.1.3 Example of a car

Figures 3, 4 and 5 show a graphical representation of an example car according to our model. This car consists of several electrical components. Normal electrical components are shown as objects in boxes with underlined text. Connections are, although they are objects as well, represented by arrows that show which components they connect. High level electrical components are shown as large boxes that contain other electrical components. Although the are represented in these diagrams they are not used in the reasoning process.

Note that the car is incomplete. It lacks, for example rear lights. The representation is based on a general notion of how cars work but it is not a (partial) representation of an existing car.

3.2 Inference knowledge

The inference model is shown in figure 6. This inference model is almost the same as the template shown in the CommonKADS book. The first difference is

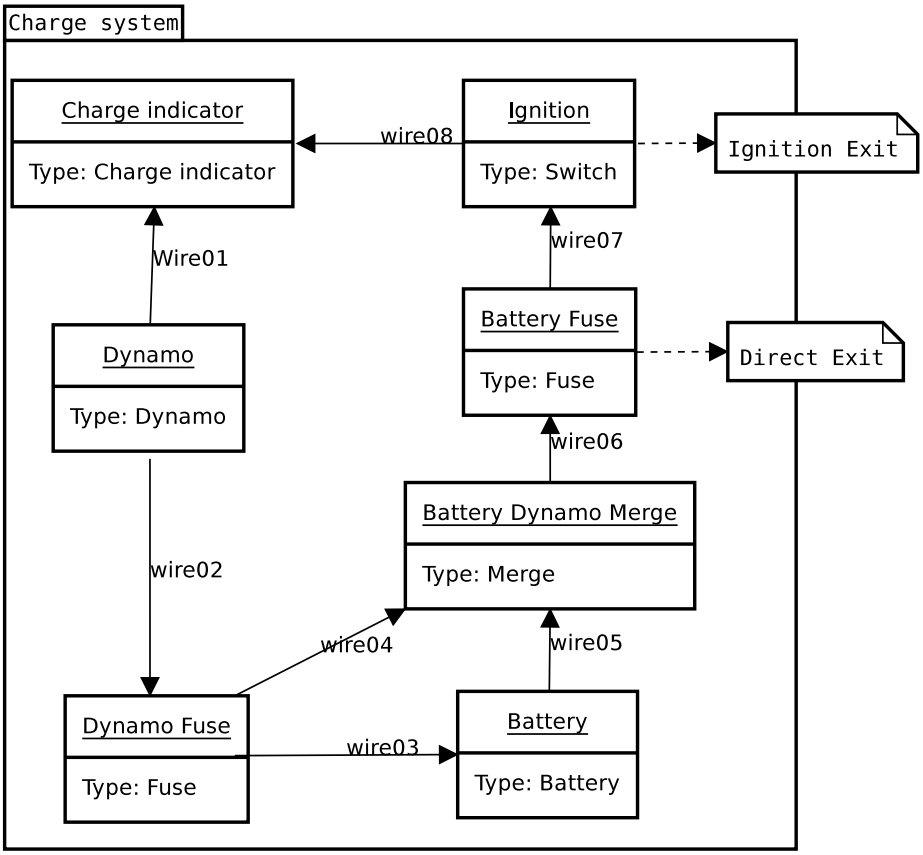

Figure 3: The charge system of our example car

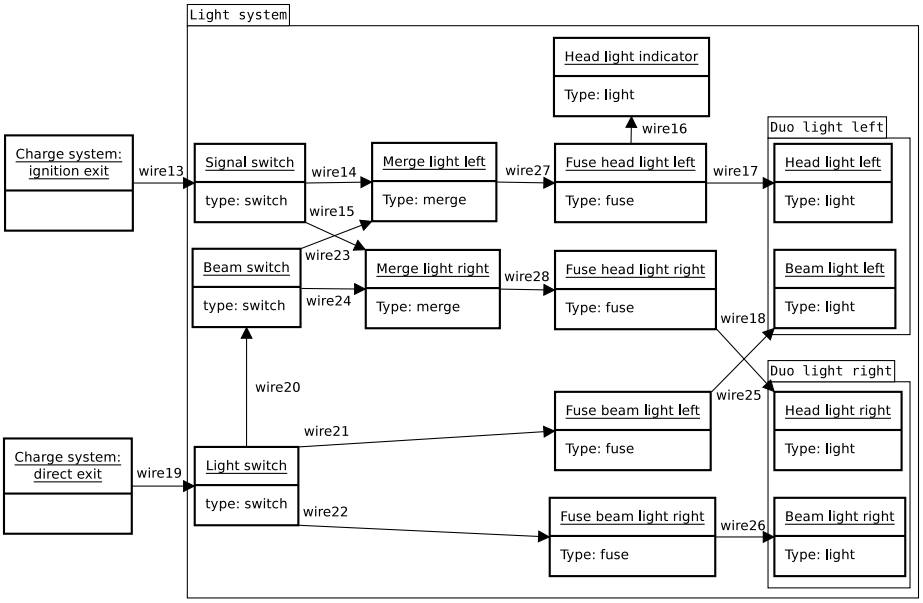

Figure 4: The light system of our example car

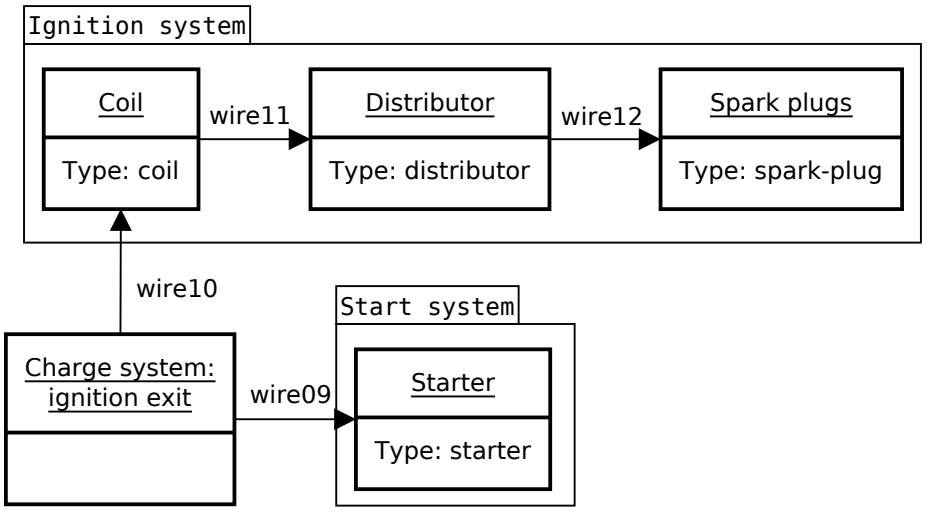

Figure 5: The start and the ignition system of our example car

an additional interaction with the user 'obtain possible selection'. This interaction was added because our user would greatly like to influence the reasoning process by proposing his own hypothesis. The second difference is the added transfer function 'try to repair'. This is added because the user has to try to repair his car to finally confirm an hypothesis. The model is annotated with a simple example of how the system could work.

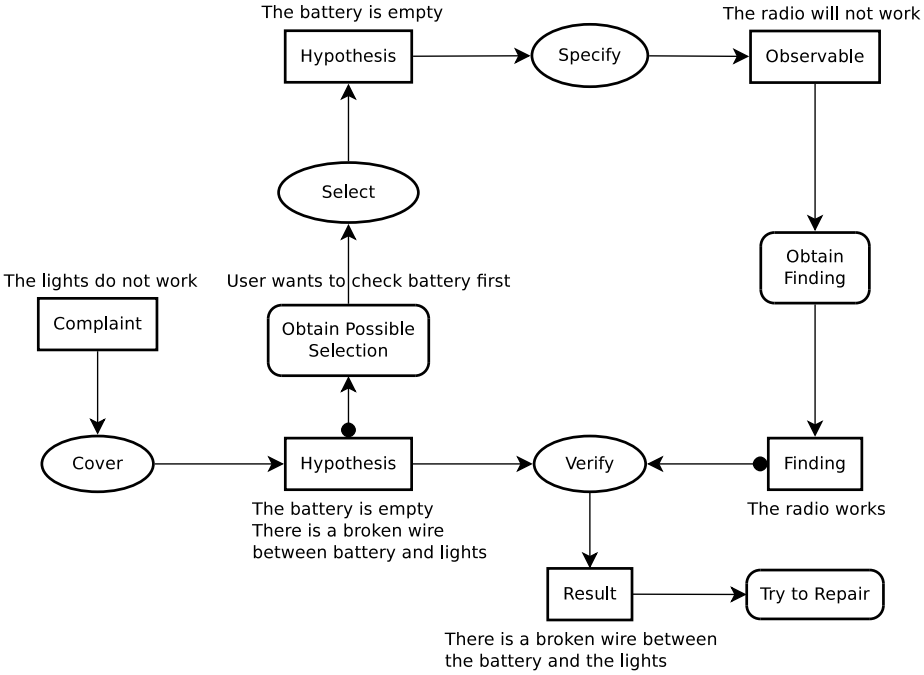

Figure 6: Annotated inference model

3.3 Task knowledge

In our context analysis we identified to major task shown in the TM-1 tables. For this project we decided to focus only on the diagnoses task. The task model of the diagnoses task is shown in figure 8. The task model shows that the diagnoses task consists of the inference and interactions of the template diagnoses task plus the 'obtain possible selection' and 'try to repair' transfer functions. 'Obtain possible selection' is used for improved user interaction about hypothesis. 'Try to repair' is used to to ask the user to repair the car and confirm or reject an hypothesis.

The task method we use to solve this task is in figure 9 and 7. Our method is inspired by the causal-covering method of CommonKADS. A major improvement is that we ask the user to repair his car directly if we cannot find any observables for an hypothesis. If we waited until it made all possible observations for all hypothesis we would focus too much on unlikely hypothesis. It's much better to ask the user to repair a likely fault.

Now wefurther specify the inferences and transfer functions.

Cover finds all components and combination of components that could possibly cause the problem. This list of hypothesis is called the differential.

Select selects a good hypothesis to test from a list of hypothesis.

Specify finds a good observable to be observed for an hypothesis.

Verify updates the differential by removing hypothesis that are not anymore possbible.

Obtain Possible Selection asks the user to select a hypothesis from the differential. The user can also keep de hypothesis found by the select inference.

Obtain Finding works with the user to do an observation of an observable and report the resulting finding.

Try to Repair asks the user to verify an hypothesis by trying to repair the car.

```
TASK diagnosis
  ROLES:
    INPUT:
      complaint: "Finding that initiates the diagnostic process";
    OUTPUT:
      faults: "The faults that could have caused the complaint";
      evidence: "The evidence gathered during diagnosis";
END TASK diagnosis;

TASK-METHOD augmented-causal-covering
  REALIZES: diagnosis
  DECOMPOSITION:
    INFERENCES: cover, select, specify, verify;
    TRANSFER-FUNCTIONS: obtain-possible-selection, obtain-finding,
        try-to-repair;
  ROLES:
    INTERMEDIATE:
      differential: "candidate solutions";
      hypothesis: "candidate solution";
      observable: "something that can be observed";
      finding: "the resulting information of an observation";
      repair-succesfull: "boolean indication result of the repair";
  CONTROL-STRUCTURE:
    cover(complaint -> differential);
    repair-succesfull := false;
    REPEAT WHILE NOT differential.size == 0 AND NOT repair-succesfull
      select(differential -> hypothesis);
      obtain-possible-selection(differential + hypothesis -> hypothesis);
      specify(hypothesis -> observable);
      IF "observables left" THEN
        obtain(observable -> finding);
        evidence := finding ADD evidence;
        verify(evidence + differential -> differential);
      ELSE
        try-to-repair(hypothesis -> repair-succesfull);
        IF NOT repair-succesfull THEN
          differential SUBSTRACT hypothesis;
        END IF
      END IF
    END WHILE
    faults := differential;
END TASK-METHOD
```

Figure 7: The task method for our diagnosis task.

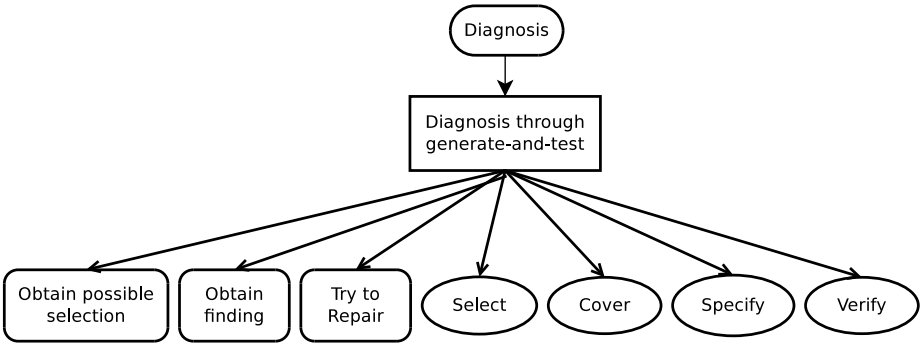

Figure 8: Task-decomposistion diagram for the diagnoses task

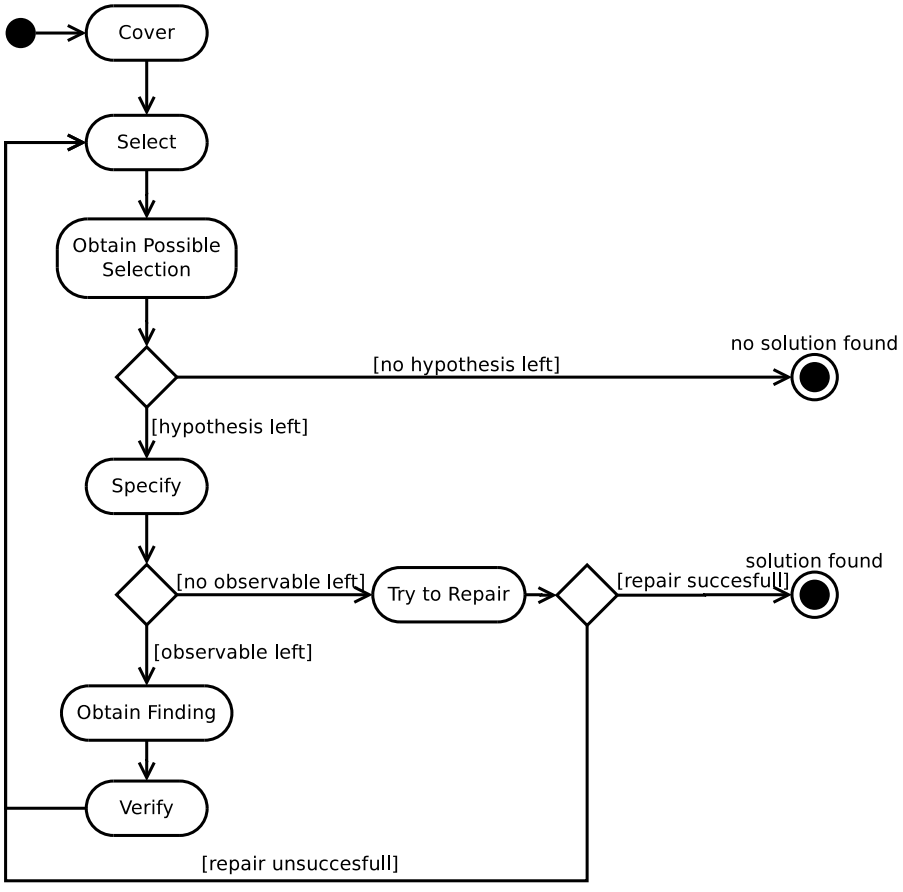

Figure 9: The task method for our diagnosis task.

4 Communication model

Since our system is highly interactive the communication model is very important. Figure 10 shows all the transaction that exist between the system and the user. Most transaction are pretty straight forward consisting of just some data being send from one to the other. The negotiate observable transaction is more complex and it therefore has its own flow control shown in figure 12. More details about these transactions are shown in tables TM-1 and TM-2.

4.1 CRA communication plan

The control flow of the communication is shown in the communication plan (figure 11). Because communication is so important in our system this plan is almost identical to the control flow of the CRA itself.

As can be seen there is the need for a complaint at the start of the plan. Once this need is satisfied the state of the system will run in circles alternating between needing hypothesis, needing observables, found covering observations and observed observation until there are either no hypothesis or no observables left. At that point the plan is finished, either successful or unsuccessful.

4.2 CRA transactions

Communication model	**Transaction Description Worksheet CM-1**
Transaction identifier/name	*Transaction 1: Report complaint*
Information object	Transferring a complaint between the *find complaint* and *cover complaint* task.
Agents involved	*Car repair assistant*: receiving the complaint; *Hobbyist*: sending the complaint
Communication plan	CRA communication plan
Constraints	Before the transaction the car repair assistant must be ready to reiceve complaints
Information exchange specification	See worksheet CM-2 below.

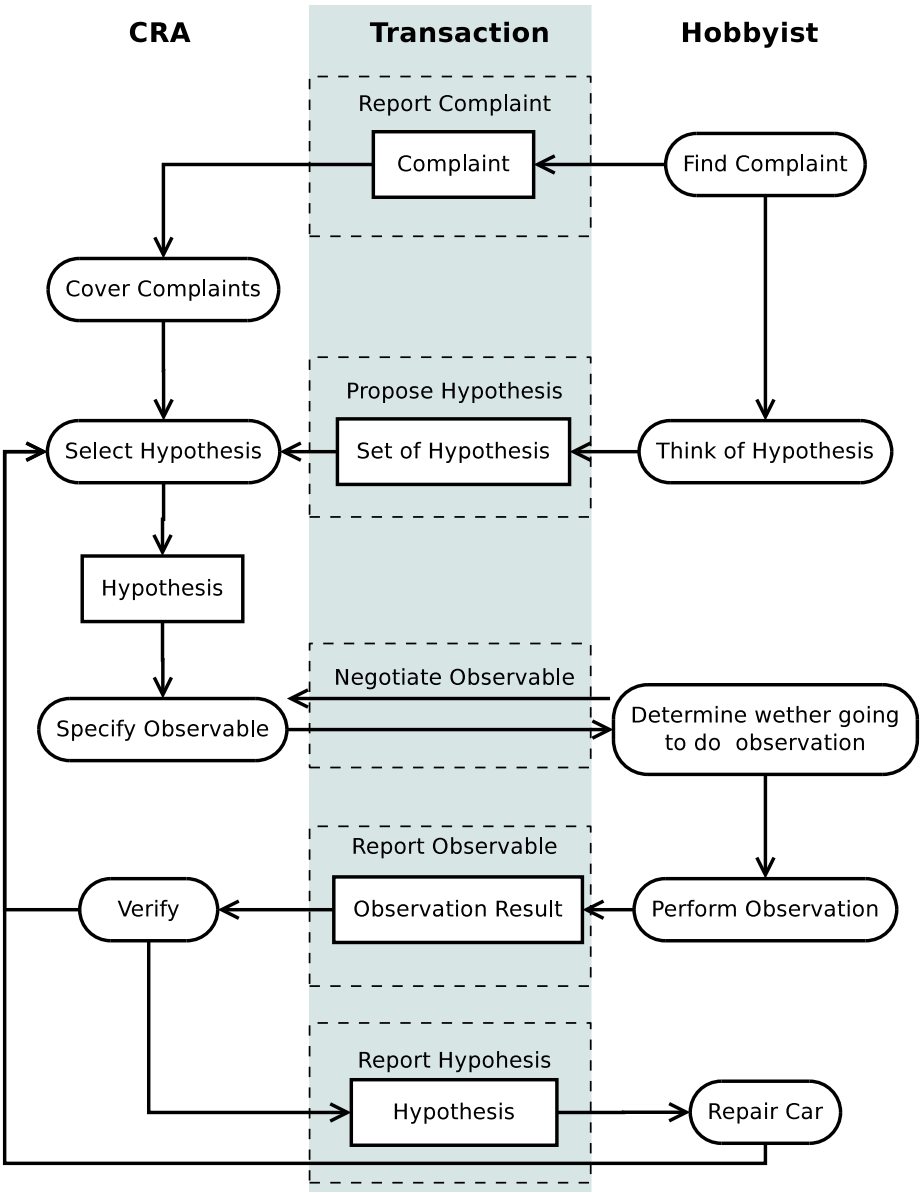

Figure 10: Dialogue diagram of the diagnoses task

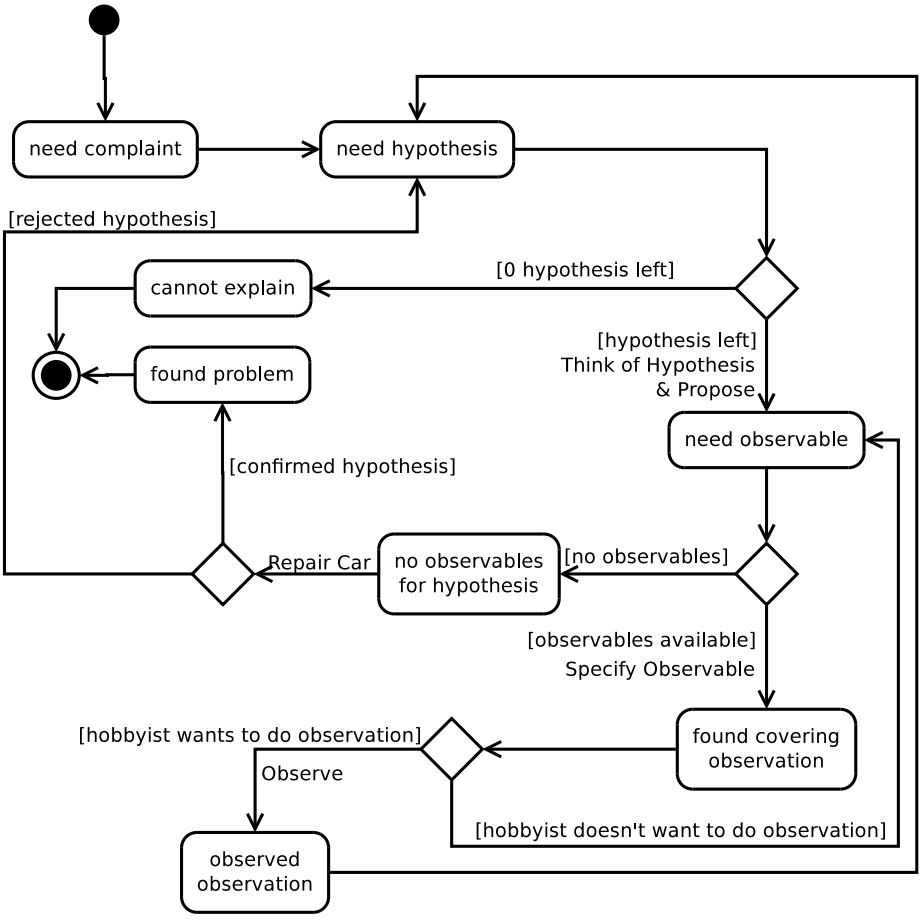

Figure 11: The communication plan of the diagnoses task.

Communication model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER/NAME	*Transaction 2: Propose hypothesis*
INFORMATION OBJECT	Transfering sets of hypothesis between the *propose hypothesis* and *select hypothesis* task.
AGENTS INVOLVED	*Hobbyist*: receiving the set of proposed hypothesis, sending a set of hypothesis (might be an empty set); *Car repair assistant*: sending a set of proposed hypothesis; receiving a set of hypothesis
COMMUNICATION PLAN	CRA communication plan
CONSTRAINTS	Before the transaction the car repair assistant must have a set of hypothesis ready. As a post condition one hypothesis has to be selected.
INFORMATION EXCHANGE SPECIFICATION	See worksheet CM-2 below.

Communication model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER/NAME	*Transaction 3: Negotiate observable*
INFORMATION OBJECT	Transferring observation instructions between the *specify observable* and *determine whether going to do observation* task.
AGENTS INVOLVED	*Car repair assistant*: sending observation instructions; *Hobbyist*: receiving observation instructions
COMMUNICATION PLAN	CRA communication plan
CONSTRAINTS	Before the transaction the car repair assistant must have a set of observables ready. As a post condition one observable must be excepted.
INFORMATION EXCHANGE SPECIFICATION	See worksheet CM-2 below.

Communication model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER/NAME	*Transaction 4: Report observable*
INFORMATION OBJECT	Transferring observation result between the *perform observation* and *verify* task.
AGENTS INVOLVED	*Hobbyist*: sending observation result; *Car repair assistant*: receiving observation result
COMMUNICATION PLAN	CRA communication plan
CONSTRAINTS	Before the transaction the hobbyist must have carried out the observation instructions and remembered there results.
INFORMATION EXCHANGE SPECIFICATION	See worksheet CM-2 below.

Communication model	**Transaction Description Worksheet CM-1**
TRANSACTION IDENTIFIER/NAME	*Transaction 5: Report hypothesis*
INFORMATION OBJECT	Transferring hypothesis result between the *verify* and *repair car* task.
AGENTS INVOLVED	*Car repair assistant*: sending hypothesis; *Hobbyist*: receiving hypothesis
COMMUNICATION PLAN	CRA communication plan
CONSTRAINTS	Before the transaction the car repair assistant must have either no observations left or he has one or less hypothesis left.
INFORMATION EXCHANGE SPECIFICATION	See worksheet CM-2 below.

Communication model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	*Transaction 1: Report complaint*
AGENTS INVOLVED	1. **Sender**(Hobbyist): Initiate repair 2. **Receiver**(Car repair assistant): Initiate repair 3. **Sender**(Car repair assistant): List of complaints 4. **Receiver**(Hobbyist): List of complaints 5. **Sender**(Hobbyist): Complaint 6. **Receiver**(Car repair assistant): Complaint
INFORMATION ITEMS	
INITIATE REPAIR	1. **Role**: A core information object. 2. **Form**: A signal indicating the start of a repair process 3. **Medium**: By starting the program using an icon or command
LIST OF COMPLAINTS	1. **Role**: A support information object. 2. **Form**: A list of strings 3. **Medium**: As menu items
COMPLAINT	1. **Role**: A core information object. 2. **Form**: An identifier 3. **Medium**: As a selection within a menu
MESSAGE SPECIFICATIONS	
INITIATION-MESSAGE	**Communication type**: ORDER **Content**: Initiate repair **From**: Hobbyist **To**: Car repair assistant
COMPLAINTS-LIST-MESSAGE	**Communication type**: REQUIRE **Content**: List of complaints and the request to chose one **From**: Car repair assistant **To**: Hobbyist
COMPLAINT-MESSAGE	**Communication type**: AGREE/REPORT **Content**: Complaint **From**: Hobbyist **To**: Car repair assistant

Communication model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	*Transaction 2: Propose hypothesis*
AGENTS INVOLVED	1. **Sender**(Car repair assistant): List of hypothesis 2. **Receiver**(Hobbyist): List of hypothesis 3. **Sender**(Hobbyist): Hypothesis 4. **Receiver**(Car repair assistant): Hypothesis
INFORMATION ITEMS	
LIST OF HYPOTHESIS	1. **Role**: A support information object. 2. **Form**: A list of strings 3. **Medium**: As menu items
HYPOTHESIS	1. **Role**: A core information object. 2. **Form**: An identifier 3. **Medium**: As a selection within a menu
MESSAGE SPECIFICATIONS	
HYPOTHESIS-LIST-MESSAGE	**Communication type**: REQUIRE **Content**: List of hypothesis and the request to chose one **From**: Car repair assistant **To**: Hobbyist
HYPOTHESIS-MESSAGE	**Communication type**: AGREE/REPORT **Content**: Hypothesis **From**: Hobbyist **To**: Car repair assistant
NO-HYPOTHESIS-MESSAGE	**Communication type**: REJECT-ta **Content**: rejection **From**: Hobbyist **To**: Car repair assistant

Communication model	**Information Exchange Specification Worksheet CM-2**
Transaction	Transaction 3: Negotiate Observable
Agents involved	1. **Sender** CRA send a request for an observation, and an explanation of that observation 2. **Receiver**: The hobbyist receives the request for an observation, and an explanation.
Information items	
	There are two information objects, the name of the observation to be done, and an explanation of that observation. 1. **Role**: The name of the observation is core, while the explanation is support. 2. **Form**: The name of the observation is a string. The explanation is canned rich text. 3. **Medium**: The name of the observation can be selected in a menu. The explanation can be shown in a text box.
Message specifications	
1. REQUEST-OBSERVATION	**Communication type**: REQUEST **Content**: Request for some observation **From**: CRA **To**: The hobbyist
2. OFFER-OBSERVATION	**Communication type**: OFFER **Content**: The Hobbyist wants to do a certain observation **From**: The hobbyist **To**: CRA
3. DO-OBSERVATION	**Communication type**: ORDER **Content**: explanation and observation the hobbyist needs to make **From**: CRA **To**: The hobbyist
4. REJECT-OBSERVATION-REQUEST	**Communication type**: REJECT-ta **Content**: Don't want to do this observation **From**: The hobbyist **To**: CRA
5. REJECT-OBSERVATION-OFFER	**Communication type**: REJECT-td **Content**: Explanation why that observation is not needed **From**: CRA **To**: The hobbyist
Control over messages	See figure 12.

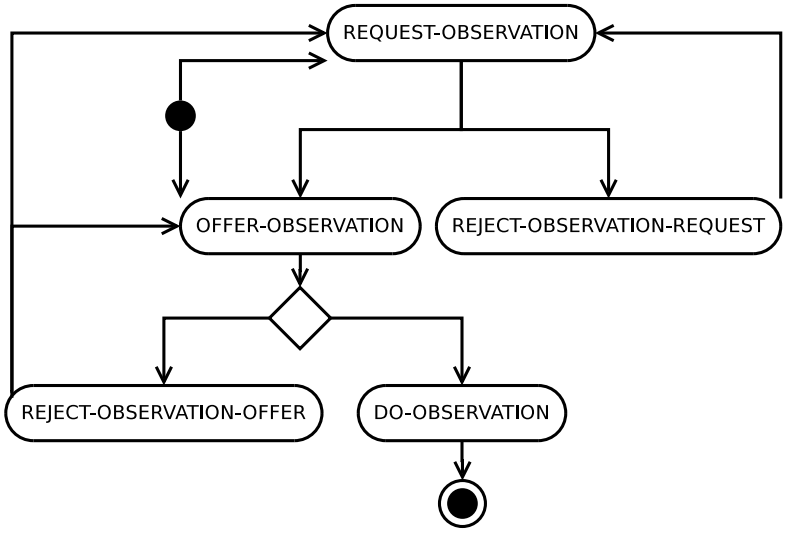

Figure 12: Control flow of the negotiate observable transaction

Communication model	**Information Exchange Specification Worksheet CM-2**
Transaction	*Transaction 4: Report observable*
Agents involved	1. **Sender**(Car repair assistant): Observation result options 2. **Receiver**(Hobbyist): Observation result options 3. **Sender**(Hobbyist): Observation result 4. **Receiver**(Car repair assistant): Observation result
Information items	
OBSERVATION RESULT OPTIONS	1. **Role**: A support information object. 2. **Form**: List of strings 3. **Medium**: Varies, it might be a menu or it might be a free form with suggestions noted separately
OBSERVATION RESULT	1. **Role**: A core information object. 2. **Form**: Varies, it might be a identifier, a number or a string. 3. **Medium**: Varies, it might be selection in a menu or it might be typed in a field.
Message specifications	
OPTION-MESSAGE	**Communication type**: ASK **Content**: Observation result options and the request to provide the actual observation result **From**: Car repair assistant **To**: Hobbyist
OBSERVATION-MESSAGE	**Communication type**: REPLY **Content**: The observation result **From**: Hobbyist **To**: Car repair assistant

Communication model	**Information Exchange Specification Worksheet CM-2**
TRANSACTION	*Transaction 5: Report hypothesis*
AGENTS INVOLVED	1. **Sender**(Car repair assistant): Hypothesis 2. **Sender**(Car repair assistant): Hypothesis argumentation 3. **Sender**(Car repair assistant): Repair plan 4. **Receiver**(Hobbyist): Hypothesis 5. **Receiver**(Hobbyist): Hypothesis argumentation 6. **Receiver**(Hobbyist): Repair plan
INFORMATION ITEMS	
HYPOTHESIS	1. **Role**: A core information object. 2. **Form**: A string stating the hypothesis 3. **Medium**: Displayed in the main window
HYPOTHESIS ARGUMENTATION	1. **Role**: A support information object. 2. **Form**: A list of strings each stating one reasoning step 3. **Medium**: Displayed in the main window
REPAIR PLAN	1. **Role**: A support information object. 2. **Form**: A text, possibly with images 3. **Medium**: Displayed in the main window
MESSAGE SPECIFICATIONS	
HYPOTHESIS-MESSAGE	**Communication type**: REPORT **Content**: The hypothesis, the hypotheses argumentation and the repair plan (plan depending on car repair information) **Reference**: car repair information **From**: Hobbyist **To**: Car repair assistant

5 Design Model

5.1 Components and Overall Structure

In this section we show the overall structure of the design. The design is going to be implemented in Java and Jess. The inference knowledge and domain knowledge is mostly in the Jess part, while the task knowledge and user interaction is in the Java part.

The modular structure can be seen in figure 13. The main application is implemented in Java with a model-view-controller design, see section 5.3. The model part starts a Jess engine and loads facts and rules that implement the system model and inferences over it.

In the working memory of the Jess engine are the domain rules, Jess facts that represent inferential knowledge of car, and the domain parts, Jess facts with knowledge about the current status of parts of the car. Domain rules are not modified over a run of the program, while domain parts are modified.

Information from the Java application goes to Jess by asserting support facts, these are then interpreted as modifications of domain parts by support rules. Information from Jess goes to the Java application by Jess queries made by that application.

The components in Jess are divided between a engine and a domain part. The domain parts consists of the domain rules and the domain parts, these are implementations of templates supplied by the engine. The engines Jess rules do forward and backward reasoning on these domain rules and domain parts, see section 5.4. The domain rules and domain parts can be substituted by rules and parts from a different domain while keeping the engine. The rules might be understandable by an expert or user, as our car repair hobbyist does. The complexity of the inferences made by the engine are not visible in the domain part.

5.2 The process flow

The overall flow of the application can be seen in the activity diagram in figure 14. It is based on the communication plan in figure 11 and the task method in figure 9.

The process start when a complaint is reported to the system. The system then covers all hypothesis that follow from the complaint. The rest of the process is one big loop. It has two exits, when a successful repair is made, and when there are no viable hypothesis left. Inside the main loop, first, a hypothesis is selected with user assistance. Secondly an attempt is made to make an observation, this is a loop where the user is asked to observe all the possible observables one after another. If the user wants do do an observation the result is used to update all hypothesis in the verify activity. If the user doesn't want do to any of the observations or there are no observables for the hypothesis, the user is asked to repair his or her car. If this repair is successfully the problem is solved and the program exits. If the repair isn't successful, the hypothesis is considered impossible by the system.

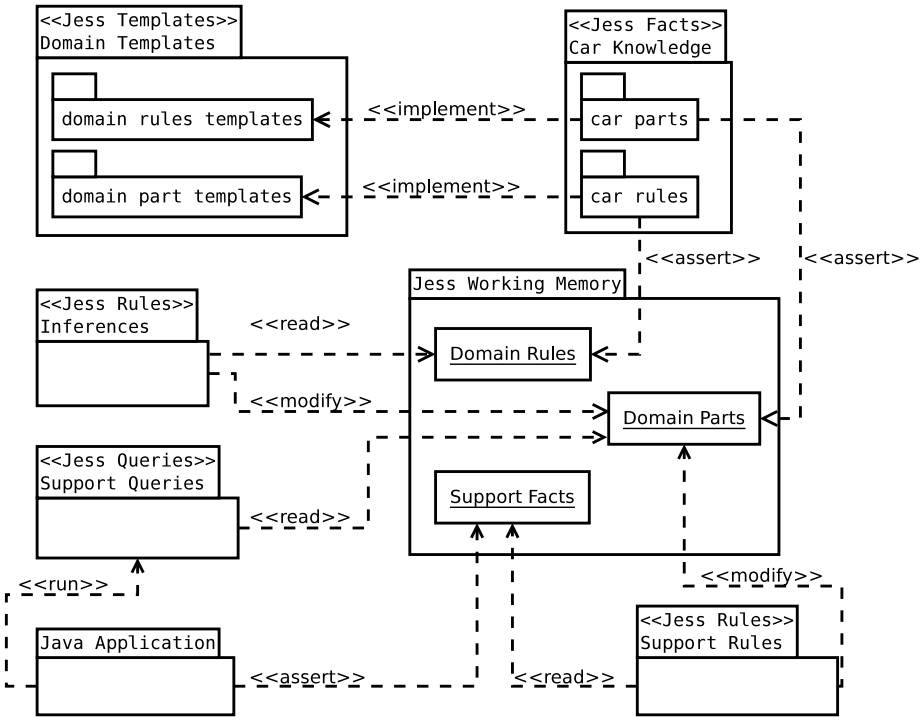

Figure 13: The modular structure of the design model

5.3 The application

The application is implemented in Java, it's class diagram is in figure 15. The model-view-controller pattern is applied. Model keeps all the inference and domain knowledge of the system. The Java class Model is partly an interface to the part of the model that is implemented in Jess and partly implements the model itself, as explained in section 5.4 and 5.1. Control manages the process flow described in the previous section, and communicates with Model and View. View implements the user interface and controls the user interaction if the model is not involved.

The Console class is used for writing and reading from the console. It's mainly used because java.io.Console is not universally supported. The Jess Engine a Jess Rete engine from the Jess library. ComponentState is a certain domain part in a (faulty) state, it wraps a identifier used in the Jess model and a pretty printable name. An hypothesis consists of one or more parts in a faulty state. The Hypothesis class is a list of ComponentStates, it also includes methods operating on a Hypothesis or a list of Hypothesis. Model delegates a considerable part of its functionality to Hypothesis. The Observable class wraps an identifier used in the Jess model and a pretty printable name. The Finding class references an Observable that has been observed and the result of that observation.

The activity diagram of figure 14 is further detailed in three sequence diagrams showing the interaction between the components Model, View and Controller.

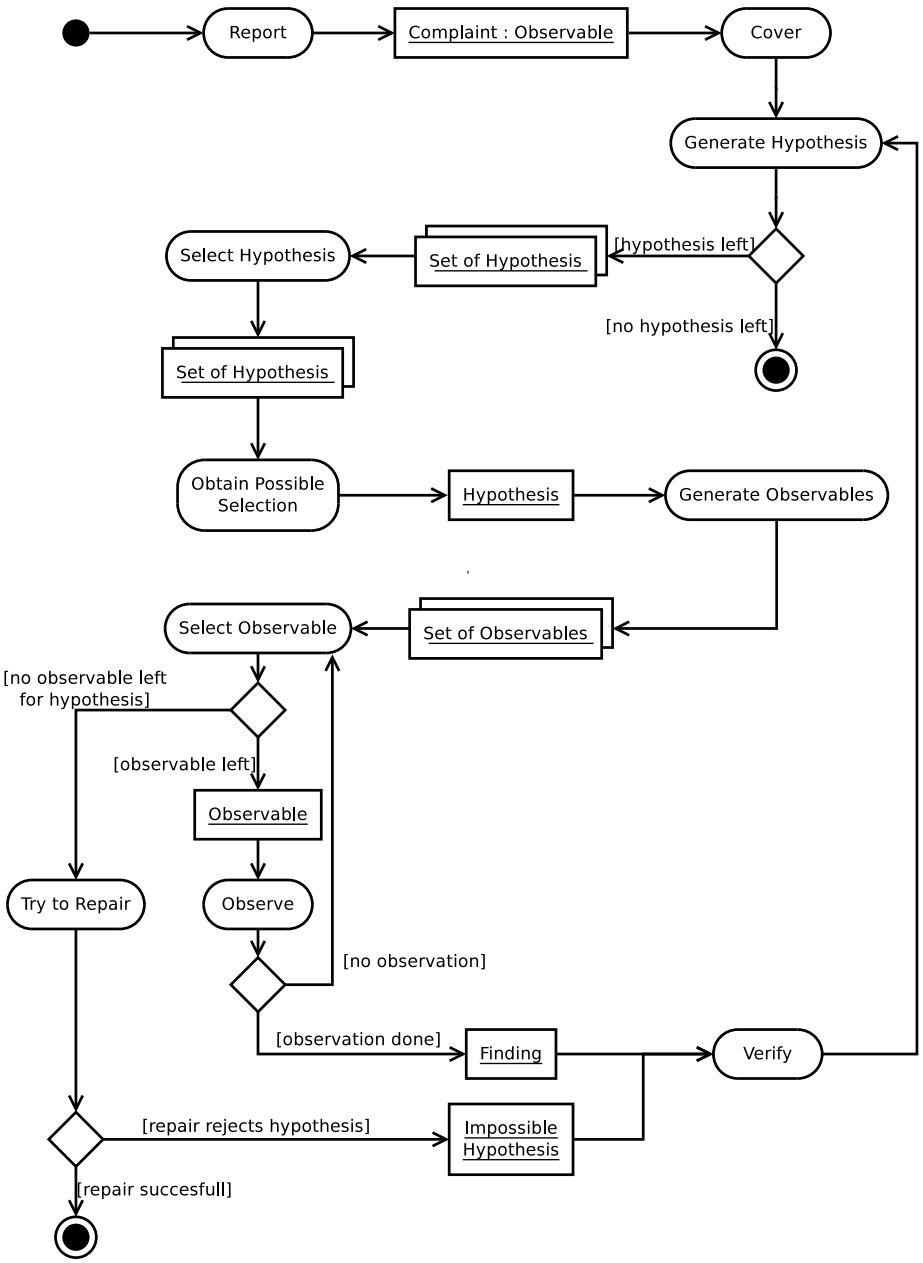

Figure 14: The overall activity diagram of the application

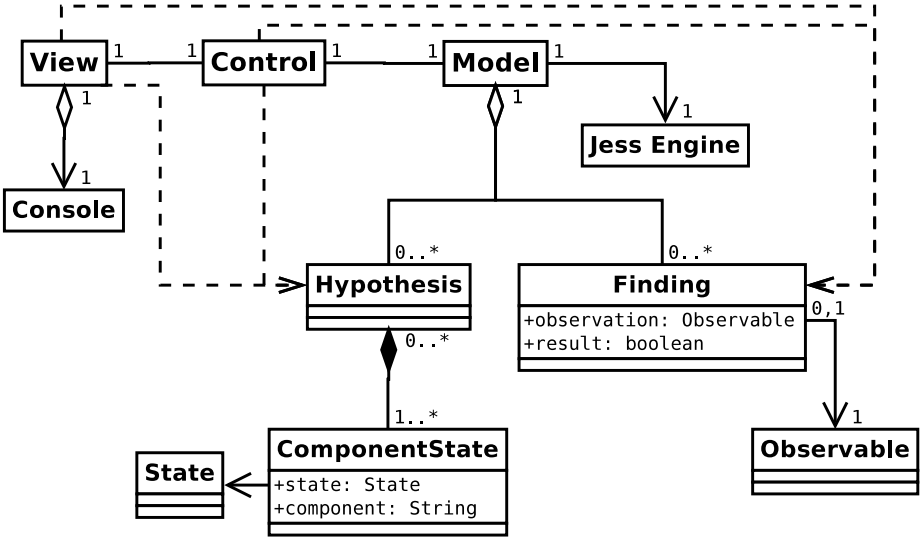

Figure 15: The class diagram of the Java application

The first diagram, in figure 16, shows how a complaint is reported and covered, it details the report and cover activities. First Model is asked for likely complaints, these complaints are shown to the user by View. The users selects an complaint from this list or enters a new one. The complain is asserted to model, which automatically covers it.

The second diagram, in figure 17, shows how hypothesis are generated and selected for testing, it details the generate hypothesis, select hypothesis and obtain possible selection activities. First Model is asked to generate all possible hypothesis. From these hypothesis one is picked as the default. Then the user is asked by View whether he wants the system to test the default hypothesis or wants to select another one, this results in a hypothesis that can be tested.

The third diagram, in figure 18, shows how observations are negotiated, attempts to repair are done and hypothesis verified by the results of these two, it details the select observable, try to repair, observe and verify activities. First a list of all possible observables for the hypothesis that is obtained from Model. In the loop the user is asked to observe one of these observables one after another, until he or she agrees to observe or there are no more observables left for the hypothesis. If the user agreed to do an observation this results in a Finding, that is asserted to Model. If there were no observable the user wants to observe for this hypothesis the user is asked to repair the car. If this repair is successful, the user is congratulated by the application and can exit. If the repair is unsuccessful it is assumed the hypothesis is false from now on, so it's asserted as impossible in Model. After all this some hypothesis might require removal, because they are not possible anymore, therefore the model is asked to verify all hypothesis.

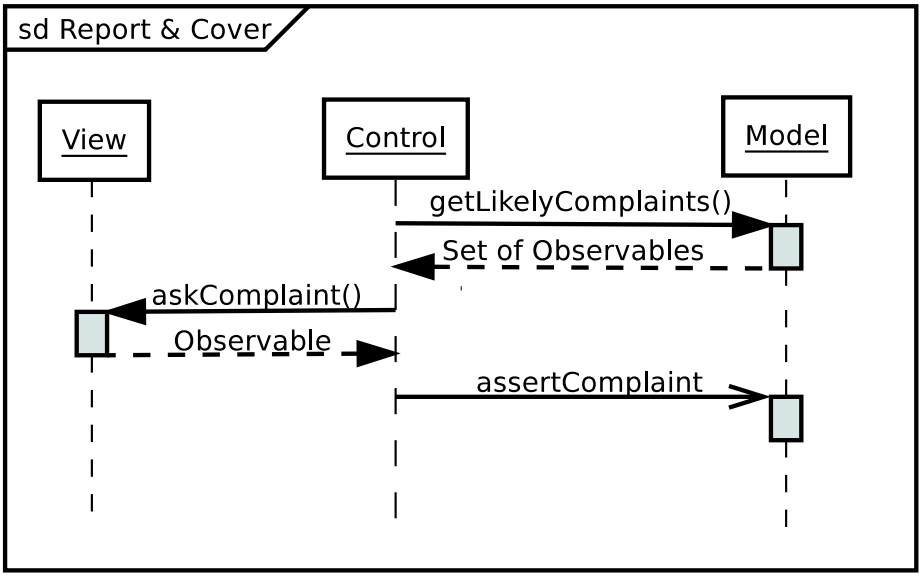

Figure 16: The sequence diagram detailing report and cover

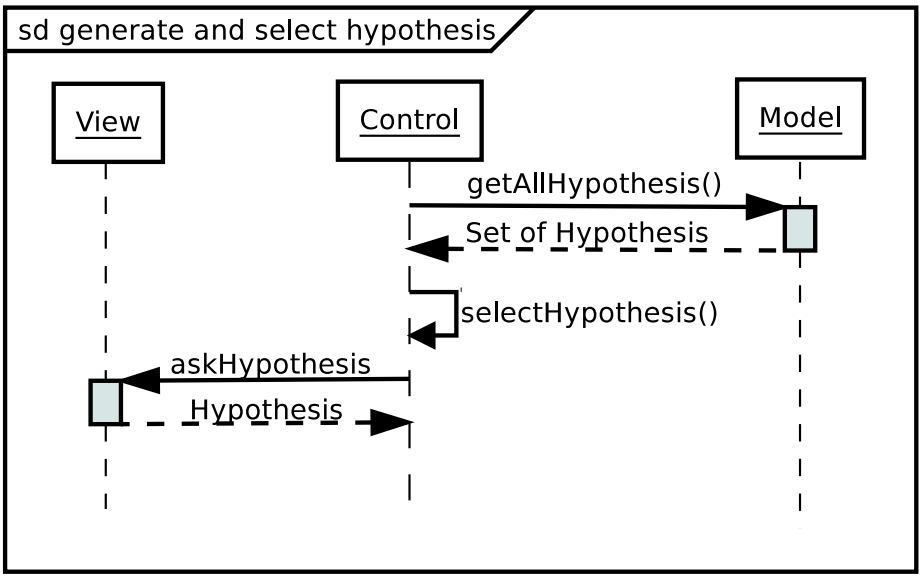

Figure 17: The sequence diagram detailing generating and selecting hypothesis

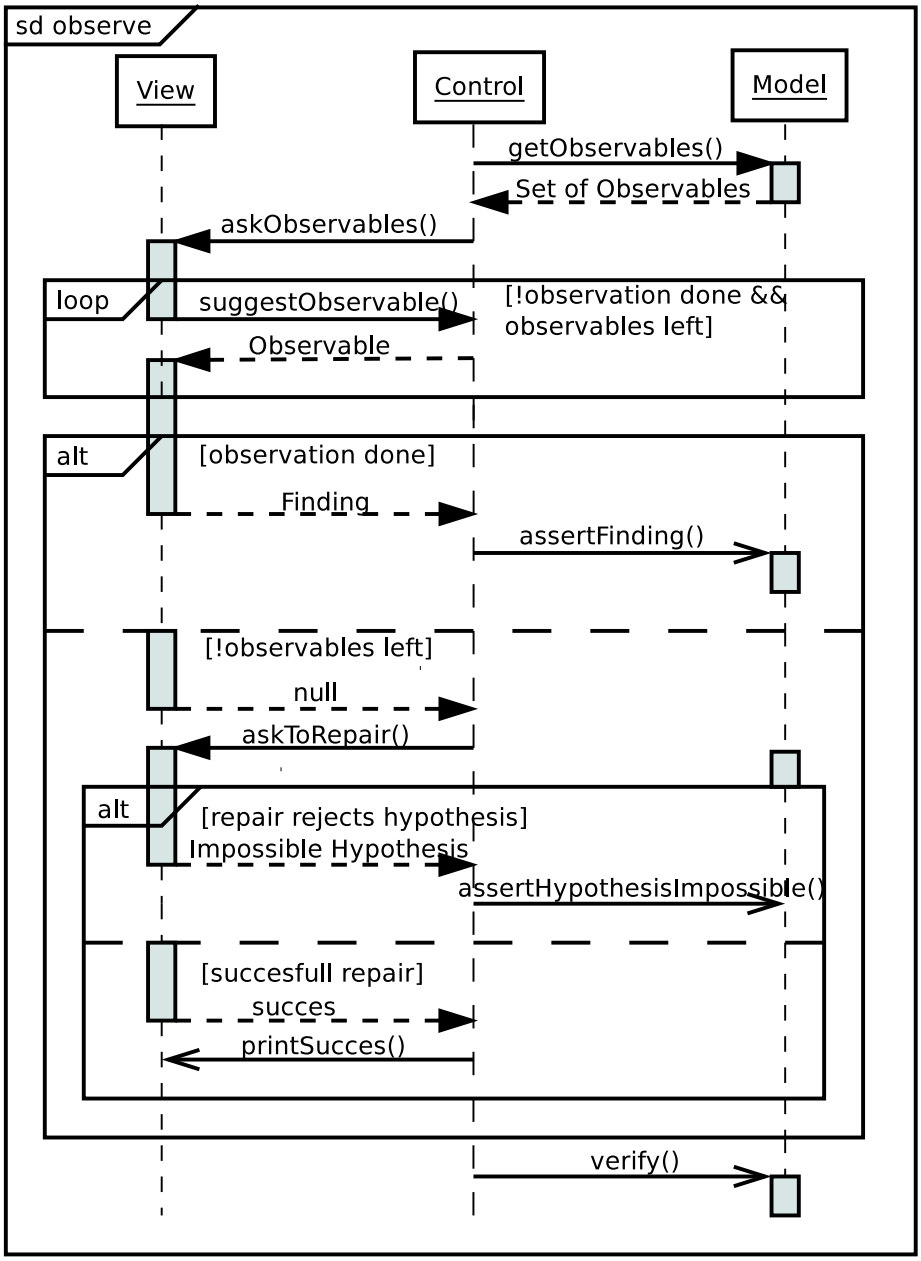

Figure 18: The sequence diagram detailing observations, try to repair and verify

5.4 Inferences

5.4.1 Knowledge base

Most inferences use the causal-model knowledge base. This knowledge base is mainly implemented in the file **car-rules.jess**. In this file there is a list of facts. Every facts implements a rule from the causal-model knowledge base. There are four types of facts: causes-to-wire, causes-from-wire, causes-from-2wires and causes-feature. These types correspond to the rule type found in the causal model knowledge base with the exception of causes-from-2wires which is a special case of the causes-from-wire rule type. Below is an example of a causes-to-wire fact.

```
(causes-to-wire
   (faulty-comp-type battery)
   (faulty-comp-state empty)
   (wire-state no-power)
)
```

This fact should be interpreted as the rule: if a component is a battery and this battery is empty then any wire connected to it will have no power. For actual reasoning these facts are used by jess rules defined in other files. This approach was chosen because it grants flexibility. For example, the addition of one rule will enable all forward reasoning for causes-to-wire. The addition of one extra rule will then enable all backward reasoning for causes-to-wire. If we implemented these rules directly as jess rules then we had to create two rules for every fact in this file.

5.4.2 Forward reasoning

The forward reasoning rules are specified in **forward-reasoning-rules.jess** and work as follows. For every fact type in **car-rules.jess** there is one rule that specifies how forward reasoning is implemented. Reasoning starts with one or more components being in a certain state. These starting states are usually a hypothesis that is being tested. Next, the direct effects of these states are derived and connected components are set to the derived state. This process then propagates until the are no more new state that can be derived. Forward reasoning is a deterministic process. Every state, or combination of states, causes only a single next state.

5.4.3 Backward reasoning

The backward reasoning rules are specified in **backward-reasoning-rules.jess**. As with the forward rules, there is a backward reasoning rule for every fact type in **car-rules.jess**, with the exception of causes-from-2wires. This fact type has two rules associated with it, one for each wire. Since the wires can be handled independently, these rules are simpler then trying to create one large rule to govern them both at the same time.

Backwards reasoning starts from an observable and propagates backward until there are no more inferences to be made. It reasons with possibilities, meaning that the observable *left head light does not work* will result in the inferences that *wire 17 might be broken* AND *wire 17 might have no power*. For

both these inferences it is then examined whether they could have been caused by one or more other malfunctions. One of the rules used in this process is shown below.

```
(defrule causes-feature-backward
"If a faulty state of a component causes an observed complaint,
that component might be in that state"
   (observable
      {observed == TRUE}
      {complaint == TRUE}
      (id ?oid)
   )
   (causes-feature
      {observable == ?oid}
      (component ?cid)
      (component-state ?cstate)
   )
   ?c <- (component
      {id == ?cid}
      (possible-states $?ps&~:(member$ ?cstate ?ps))
   )
   =>
   (modify ?c
      (possible-states (create$ ?ps ?cstate))
   )
)
```

The rule above has four parts. The first part triggers when there is an observable that was marked as a complaint. The second part triggers when there exists a 'rule' (remember that these rules are implemented as jess facts) that links this observable with a component and a state. The third part checks whether this component already has this state as a possible or impossible state. The fourth part is the consequent and adds the state to the possible-states of the component.

The other rules will propagate possible states in a similar manner such that, at the end of the reasoning process, every component that could potentially be the cause of the complaint has a possible state that would cause this component to malfunction.

5.4.4 Cover

The cover step has been divided into two parts. The first part creates a list of single component hypothesis, which we will call *basic hypothesis*. The second part expands some of these basic hypothesis into multi component hypothesis, which we will call *composed hypothesis*.

The first part is done by marking an observable as a complaint, done via the support rule in **support.jess**, and then to run the backward reasoning. When this process is finished, a query will collect all component-state combinations and each of these combinations becomes a hypothesis.

The resulting set of hypothesis still has one problem, there might be basic hypothesis in this set that do not cause the inital complaint by themselves.

The reason for this is that there could have been a merge in the backward reasoning trace. A merge is a component that has two inputs and that gives power when one of its inputs provides power, functioning as a logical OR. If such a component has no power then this is because there are two components that do not provide power two the merge. As such, the hypothesis should now involve two components. Because of implementation problems this issue has not been addressed in jess. Instead it has been solved in the java code and it comprises the second part of cover.

The second part of cover expands the basic hypothesis when the hypothesis alone is not enough to explain the complaint. This part has been coded in the java function List and is shown below.

```
private List<Hypothesis>
        expandHypothesis(List<Hypothesis> allHypothesis) {
    ...

    for(int i=0; i<allHypothesis.size(); i++){
        currentHypothesis = allHypothesis.get(i);

        if(currentHypothesis.directCause()){
            result.add(allHypothesis.get(i));
        } else {
            currentHypothesis.maxIndex = basicHypothesis.size();
            basicHypothesis.add(currentHypothesis);
            candidateHypothesis.add(currentHypothesis);
        }
    }
    ...
```

The first part of the List function creates a list containing those hypothesis that should be expanded, namely those that would not by themselves cause the complaint. Those that cause the complaint by themselves are already complete hypothesis and are immediately added to the result.

To check whether an hypothesis causes a complaint on its own forward reasoning is used. When the function currentHypothesis.directCause() is called, the currentHypothesis is asserted in jess, forward reasoning is performed, and a support rule then checks whether the complaint was reached. The result of this reasoning proces is then returned.

The hypothesis that are selected for expansion are added to two different lists: the basic hypothesis list and the candidate hypothesis list. The basic hypothesis list will be used to expand other hypothesis. The candidate hypothesis list is the list of hypothesis to be expanded and it will grow when hypothesis need to be expanded more then once.

```
    ...
    //Expand the candidate hypothesis
    while(candidateHypothesis.peek() != null){
        //Get the first hypothesis
        currentHypothesis = candidateHypothesis.pop();
        for(int j=currentHypothesis.maxIndex; j<basicHypothesis.size(); ↩
        j++){
            //For each basic hypothesis that we have not tried with this↩
             combination
```

```
                //Create a new composed hypothesis by expanding the ↩
                candidate with the basic hypothesis
                newHypothesis = currentHypothesis.clone();
                newHypothesis.add(basicHypothesis.get(j));

                if(newHypothesis.directCause()){
                    //If the new hypothesis is a direct cause then a new ↩
                    composed hypothesis has been found
                    //Add it to the result
                    result.add(newHypothesis);
                } else {
                    //If the new hypothesis is not a direct cause then it ↩
                    might need further expanding.
                    //Check whether new information was gained with the ↩
                    composed hypothesis by
                    //checking whether the new hypothesis causes more state ↩
                    changes then
                    //the two previous hypothesis combined
                    if((currentHypothesis.nrStateChanges() + basicHypothesis↩
                    .get(j).nrStateChanges() < newHypothesis.nrStateChanges↩
                    ())){
                        //If new information was gained then the new ↩
                        hypothesis
                        //should get further expanded
                        candidateHypothesis.add(newHypothesis);
                    }
                }
            }
        }

        return result;
    }
```

5.4.5 Select

The select step determines which hypothesis from the list generated in cover will be suggested to the user. At this point, select will always suggest the last hypothesis of the list and no reasoning is performed.

5.4.6 Specify

Once an hypothesis is selected the observables, that could be used to falsify the hypothesis, are generated in the specify step. For this task, the complete hypothesis is asserted in the jess reasoning engine. A support rule modifies the states of the components to match the hypothesis, and then the forward reasoning is applied. After this, a query is used to collect the observables that are affected by this hypothesis.

5.4.7 Verify

When a new observation is made by the user, and the results of this observation are communicated back to the car repair assistant, all hypothesis are tested to check whether they are consistent with the new observation. Thus, each

hypothesis in our set of possible hypothesis is asserted to the jess rule engine, the forward reasoning is applied, and a query returns whether there was a contradiction caused by this hypothesis. In this manner, every hypothesis that causes a contradiction is removed.

Note that the hypothesis that was being tested might be unaffected. In this case, the hypothesis remains in the list of possible hypothesis and it could be examined again.

6 Conclusion

6.1 The process

The process of creating the knowledge based system went well in general.

There was a problem with, unsurprisingly, the contact with our expert. Although the expert, a student car technician, was chosen because he was more easily accessible then the average car mechanic it was still difficult to get an appointment. Because of this the initial domain schema was eventually created without the help of the expert. Also, since we initially waited for an appointment there was a great delay in our project. The information that was eventually obtained through our expert could be added rather nicely to our domain knowledge, but this ordering undoubtedly influenced the final model.

We used most of the models and methods of CommonKADS during the project. The organization and agent model is useful to define the boundary and context of the system. As the environment of the system is very small, it only works together with the car hobbyist, we quickly finished those. With the task model we could better define and narrow down the task of the system. The knowledge model and the communication model are very useful. They form the foundation of our implementation.

The knowledge model makes clear what rules and inferences should be used by the system. It also created problems which have disappeared when we implemented it. We struggled with the difference between inferences and rules types, designing a rule type for the cover inference separately from a rule type for the verify and specify step. This proved unnecessary when we implemented the rules.

The CRA needs to work together with the car hobbyist. Communication is essential for the system. The communication model is great to model this. Creating this model was time-consuming, but proved worthwhile.

On a first attempt we did try to make a design model based on the model view controller paradigm. It was very unclear how this should be done. With little time left and good preceding models we decided to just start the implementation based on the knowledge and communication model. We implemented the causal rules and component knowledge from the knowledge model in Jess. These rules were used in the inferences that we also implemented in Jess. The control of the program is based on the communication plan and dialog diagram of the communication model and implemented in Java.

On a second attempt we documented what we implemented. From the activity diagram and sequence diagrams in de design model a model view controller patern emerged. Now it was clear how the design should proceed. Based on this new design we changed out program. Classes for hypothesis, components, states, observables and findings were added. The single application was split up between a controller, a view and a model class. At first this enabled us to add the try to repair step in the design and program. Second it allowed us move the logic to deal with hypothesis with multiple faulty components from Jess to the Java class Hypothesis, leading to improvements in dealing with multiple hypothesis. We didn't need to change the core logic of the program. Third it enables improvements in the user interaction.

6.2 The final result

The Car Repair Assistant performs its tasks reasonably well. It gives all possible causes and it will slowly reduce the number of possible causes by querying the user about various observables. The user can also steers this process by giving suggestion about which causes should be examined next. The system is also very flexible and using it with an other car, or even an other electrical system is very simple. Nevertheless, there are a lot of possible improvements.

An other important point that could be improved upon is the method that the CRA uses to suggest hypothesis and observables. Right now it just choses one with static preferences. It preferes hypothesis without wires and observables that don't observe the current hypothesis. It would be a lot better if these would be selected based upon the ease of obtaining the observations or based upon the information gain.

A last obvious point of improvement would be in the interface. This system would be much more practical if it worked with a transparent picture of a car that depicted all the components and their locations. This would make the tool a lot easier to use for the users.

Unfortunately there was little time to create the implementation of the final Car Repair Assistant. This is part due to the delay at the start of our project and part due to the fact that two months is not very long to create a fully functional knowledge system. The extra time provided to us to improve our work enables us to improve the design model and documentation, and to improve the visible structure of the program to confirm with the design.