Intra-vehicular verification and control: a two-pronged approach

Atif Alvi* and Zubair Nabi

Department of Computer Science,
Lahore University of Management Sciences,
Lahore 54792, Pakistan
Fax:+92 042 3589 8315
Email: atif.alvi@lums.edu.pk
Email: zubair.nabi@lums.edu.pk
*Corresponding author

David J. Greaves

Computer Laboratory,
University of Cambridge,
Cambridge CB3 0FD, UK
Fax: +44 0 1223 334678
Email: david.greaves@cl.cam.ac.uk

Rashid Mehmood

School of Engineering,
University of Swansea,
Swansea SA2 8PP, UK
Fax: +44 01792 295676
Email: r.mehmood@gmail.com

Abstract: Modern vehicles are equipped with hundreds of embedded networked components with computational, sensory and actuation powers. Reliable functioning and interaction of these components are vital for the safety of the vehicle and its passengers. We present an architecture that deals with the intra-vehicular network at both component and system levels. At the component level, our technique formally verifies compatibility of each component with the rest of the system. At the system level, we provide means to define overall behaviour by using first-order logic rules in an ontological space. Overall, we eliminate the hazards associated with integrating heterogeneous components in a car network domain and enable a knowledgeable user to define network behaviour easily.

Keywords: vehicular network; pervasive computing; real-time systems; ontology; rule-based control; verification; feature interaction.

Reference to this paper should be made as follows: Alvi, A., Nabi, Z., Greaves, D.J. and Mehmood, R. (2011) 'Intra-vehicular verification and control: a two-pronged approach', *Int. J. Vehicle Information and Communication Systems*, Vol. 2, Nos. 3/4, pp.248–268.

Biographical notes: Atif Alvi is an Assistant Professor in the Department of Computer Science at the Lahore University of Management Science (LUMS). He did his BSc in Electrical Engineering with honours from UET, Lahore, his MSc in Computer Science from LUMS and his PhD, in 2008, from the Computer Laboratory at Cambridge University. After Bachelor's degree, he served as Instrumentation and Control Engineer at NESPAK, the premier engineering services firm in Pakistan. Subsequently, he was also Manager Operations at an innovative network services firm. He has trained public university IT teachers and provided consultancy services to various institutions. He is the co-author of the Computer Science textbook for grade XII in the provincial public schools. He is interested in controlling the behaviour and detecting conflicts in pervasive domains using knowledge bases and rules. He has applied this approach to home and campus domains and is now researching inter- and intra-vehicular network domains.

Zubair Nabi is a Master's student in the Department of Computer Science at the Lahore University of Management Sciences where he is also both a Research Assistant and a Teaching Assistant. His research interests include data intensive computing, virtualisation and cloud computing.

David J. Greaves is a Senior Lecturer in Computer Science at the Computer Laboratory, University of Cambridge and a Fellow of Corpus Christi College. He is a member of the Systems Research Group and Processor Architecture Groups. He also participates in the Hardware Verification Group. He undertakes research in the area of hardware design and system specification with emphasis on component interconnection. Recently, has been working on tools for advanced hardware description using logic programming and automated decision procedures and is now applying these techniques to more general software system specification and automated assembly.

Rashid Mehmood currently holds a Lectureship in Computing and Communications at the College of Engineering, Swansea University. He leads a research group at Swansea that is developing state of the art technologies through cross-disciplinary research and international collaborations. He has a multi-disciplinary background with qualifications in Electronics and Communications Engineering, Business and Computing from institutions including Universities of Oxford, Birmingham and Cambridge. His core expertise is in stochastic modelling, simulation and analysis of large-scale datasets and complex systems, and high performance computing. He has applied his expertise in a diverse range of application areas including telecommunications, intelligent transportation systems and healthcare. He has led and contributed to a large number of academic-industrial collaborative projects including multiple DTI/TSB academia-industry collaborative project and six UK-wide Research Councils Digital Economy research clusters. He is a member of ACM, OSA and IET, and a Senior Member of IEEE. He is the Second Vice-Chairman of the IET Wales South West Network.

This paper is a revised and expanded version of a paper entitled 'Intra-vehicular verification and control: a two-pronged approach' presented at the '2nd International Workshop on Communication Technologies for Vehicles, Nets4Cars'2010 (affiliated with the 7th IEEE CSNDSP)', Northumbria University, Newcastle, UK, 21–23 July 2010.

1 Introduction

Vehicles today hide enormous complexity under the hood. A veritable universe of heterogeneous components inside must function in the most reliable and smooth way to meet the stringent real-time requirements of safety and passenger satisfaction. Until now, most of the research in the automotive domain has been geared towards either improvising *ad hoc* inter-vehicular networks for safety, infotainment, etc. (Hsu and Chen, 2005; Yousefi et al., 2006), or optimising the intra-vehicular network (autosar; Berwanger et al., 2001; CAN, 1992). Not as much attention has been paid to the problem of static and dynamic verification of intra-vehicle network's components. Such verification is essential to ensure that there is no feature interaction or conflicting behaviour between components. Also, there is a need to enable the driver and passengers to control their ambient environment in a more customisable way than is presently possible.

The vision of pervasive computing is to have a wealth of embedded processors and applications serving user needs in an imperceptible way (Weiser, 1999). To realise this vision, there must exist complete harmony and interoperability among all the components (Edwards and Grinter, 2001). The intelligent vehicular *ad hoc* networks (InVANETs) can be categorised as enablers of a pervasive environment where the vehicles communicate with each other and with roadside equipment to facilitate passengers through applications ranging from traffic congestion control to commerce. Similarly, the intra-vehicular network of embedded processors, sensors and other nodes can be thought of as a smaller universe in the immediate vicinity of the passengers, comprising applications and functions that envelope the passengers in a pervasive computing environment.

Generally, the closed domain of an automobile is like any other intelligent environment with a boundary (physical or logical), e.g. a smart home, a railway carriage or an industrial plant. It is a microcosm of a pervasive environment where concurrent applications running in electronic control units (ECUs) share many common sensors, actuators and feedback paths through the physical part of the domain, while having to abide by common, basic liveness and consistency rules to ensure proper operation of the vehicular domain.

The term *feature interaction* has its roots in the telecommunications field and was coined by Bellcore in the early 80s (Keck and Kuehn, 1998). We can think of a feature as a unit of functionality – a facet of behaviour – of a system. Some feature interactions are harmless but the majority is severely damaging to system development and user expectations. It is a major research challenge to predict, identify and resolve feature interactions. Cameron et al. (1994) is the seminal paper for telephony interactions and lists many examples. Bowen et al. (1989) give the first framework for studying the feature interaction problem.

There are many other areas where the feature interaction problem has been studied. Turner et al. (1999) make a case for recognising features as first-class objects in the software process and introduce the area of *feature engineering* to incorporate their role in a broad range of software life-cycle activities. According to Calder and Magill (2000):

> The subject [of feature interaction] has relevance to any domain where separate software entities control a shared resource.

Hence, pervasive computing environments like a vehicular network certainly represent domains where feature interaction is likely to take place. This means that the functionality of different nodes in the car network represent features and have the potential to interact in a conflicting manner.

Our previous work (Alvi and Greaves, 2006; Alvi and Greaves, 2008; Greaves et al., 2008) has focused on using a home domain as a testbed for our techniques. There, we successfully implemented a prototype system that controlled the various devices and appliances in a reliable, conflict-free and easily configurable manner. We have also developed data structures, algorithms, and parallel computing methods for stochastic modelling and (probabilistic) model checking of large-scale systems using different high-level formalisms, e.g. see Mehmood (2004); Mehmood and Crowcroft (2005). We can use these techniques in conjunction to provide more flexible and complete low-level and high-level control of large-scale pervasive domains.

We should note that existing methodologies for safety critical software, like Ravenscar Profile (Burns, 1999) for Ada and other coding standards, are good at avoiding certain types of coding error, such as dangling pointers, overflow and infinite loops, but a totally different approach is needed to handle issues of system-level concurrency. Another relevant software architecture is the 4D-RCS Reference Model Architecture for military unmanned vehicles (Albus et al., 2002).

The recent automotive standardisation effort AUTOSAR (automotive open system architecture) autosar by manufacturers, vendors and tool developers aims to define a common E/E (electrical/electronic) component-based software architecture to facilitate scalability, integration, transferability and maintainability. AUTOSAR has safety as one of its prime goals and invites novel techniques to fulfil the stringent safety requirements of automotive applications. To enable feature interaction checks in AUTOSAR, we augment it with ontologies and *rule bundles.* At design time, when ECUs are configured through the AUTOSAR pipeline, detection of any feature interaction terminates the process.

The rest of the paper is organised as follows. Section 2 introduces the two-pronged approach. An enhancement to the AUTOSAR standard that focuses on a check for feature interaction at design time is given in Section 3. We give an overview of location-based-services (LBS) in the vehicular domain in Section 4. Advantages of the two-pronged approach are listed in Section 5 and related work is outlined in Section 6. Section 7 concludes this paper.

2 The two-pronged approach

2.1 Overview of architecture

Our control and verification model of the vehicle's network is operative at two levels: component and system (Figure 1). The component level design ensures that all the network components (microcontrollers, sensors and actuators) are formally verified to cause no feature interaction with other components and to obey some standing rules defined by the designers of the system. An ontology substrate is used to hold the attributes of the system to perform various types of reasoning over them (consistency checking and classification). Higher level rules in first-order logic authored by the designers and the owner of the car form the system level.

2.2 Component level verification and control

In our approach, space is divided into domains and components are prevented from sending commands over the network in the vehicular domain until their internal applications have been validated by a Domain Manager. To do this, they offer an *application digest* which is a description of their active behaviour, so that automated reasoning techniques, like model checking, can be run before granting an application code bundle the right to send commands (Greaves et al., 2008). For example, the accelerator pedal sensor cannot send commands to the fuel intake valve without such vetting beforehand at design time.

As illustrated in Figure 2, we can represent the car network as a *domain of participation* (DoP), demarcated by the dotted line, comprising sensors, actuators, ECUs, application bundles and an ontological car model. A DoP corresponds to a community of participating applications that are sharing resources and which may interact in various intended and unintended ways, potentially causing feature interaction. A simple DoP is typically a physical space, such as a vehicle, house or factory. The Domain Manager uses a summary of the behaviour of each component to check whether the components are compatible. Each component may also offer assertions that it wishes to be held in the networked domain. There are a number of standing rules of the domain that ensure safety and liveness and prohibit command conflicts. The ontological car model gets updated dynamically as a result of changes in the state of the car domain. This happens due to the arriving and departing components and the functioning of different nodes in the domain, such as sensors and actuators.

There are many potential forms of application digest: in previous work, we investigated reflecting the key behaviour of the components using our Pushlogic language (Alvi and Greaves, 2006) and Common Intermediate Language bytecode used by the .NET Framework and Mono[1] (Greaves et al., 2008). Any popular bytecode format for which a variety of compilers exist can be chosen. Figure 3 shows how the application bundle from one of the components of the car network written in .NET bytecode for a generic but standard car network ontology is first *rehydrated*, i.e. bound. It is bound by the Domain Manager to the specific car manufacturer and model by using the resident ontology. It is then checked against the standing rules for the car network and against the code of other applications to detect any feature interaction. A successful completion of the verification step allows that component to be functional. Otherwise, it is rejected.

The car domain is dynamic in the number of components as we replace and add them during the lifetime of the car, e.g. engine parts, on-board global positioning system (GPS), navigation system and music system. This means that there will be continuous on-the-fly verification of application digests for inter-compatibility, in addition to design time verification. As an example, consider the dynamic docking of a music player with the on-steering-column volume and transport controls. In this case, dynamic checking at the time of the binding seems sensible. The required functionality should not, in any way, be compromised by such dynamic verifications and bindings. For instance, the traffic announcement messages on the radio must still be able to pause the playing track. Hence, there is a need to adopt and develop efficient offline and online verification techniques.

Figure 1 Two levels of car network control (see online version for colours)

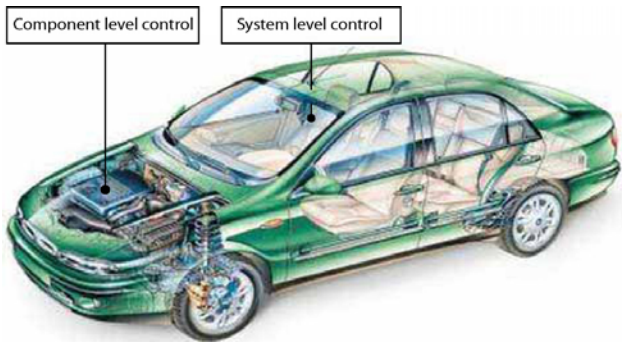


Figure 2 DoP representation of car network

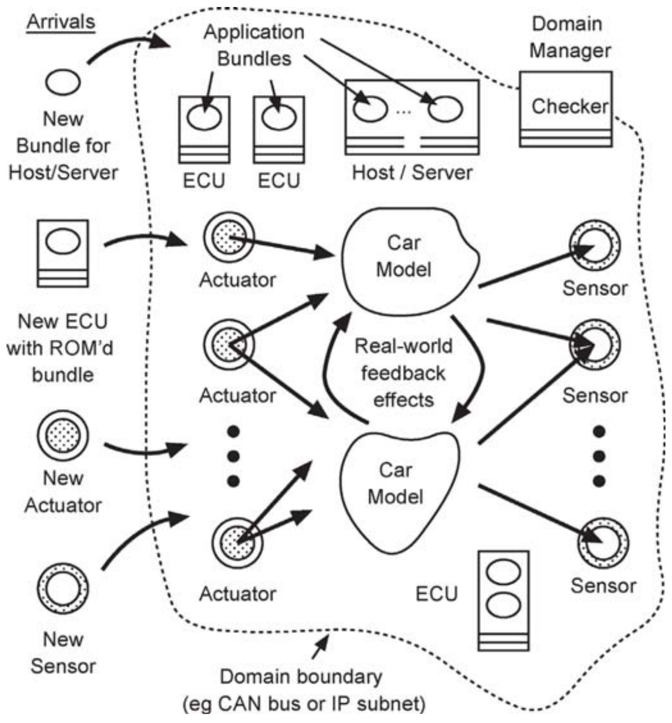
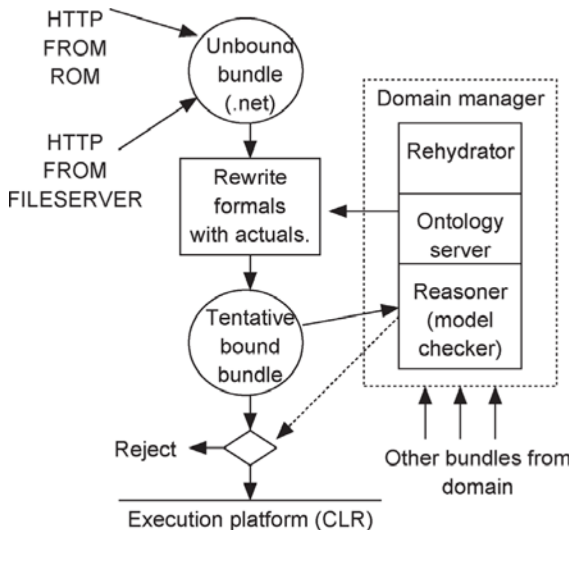


Figure 3 Binding and checking application bundles



Our repository of knowledge about the car network domain is in the form of an ontology. Both component level and system level verification and control mechanisms make use of the same ontological database, which we describe next.

2.3 Ontology knowledge base

We make use of an ontology knowledge base to both capture the properties and state of the vehicular network and apply rules to constrain its behaviour. This enables the car manufacturer and the car owner to further define the behaviour of the components and the composite services at a system level of abstraction. The lower component level verifications also use the same ontological database, which is populated at design time by the manufacturer. Hence, the safety and liveness assertions range over terms and predicates that are defined by the machine-readable ontology.

In a recent definition by Gruber[2] in Liu and Özsu (2009), ontology is defined in the context of information and computer sciences as:

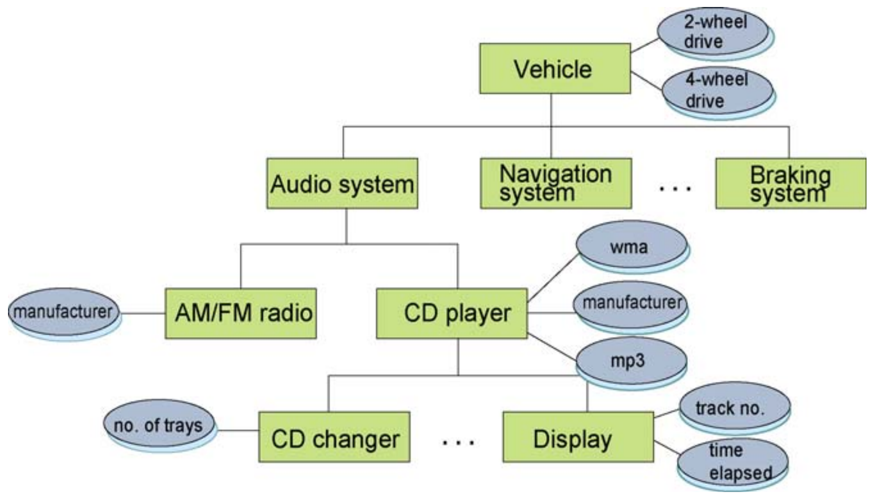
> An ontology defines a set of representational primitives with which to model a domain of knowledge or discourse.

The representational primitives are typically classes (or sets), attributes and relationships among class members.

Ontologies are at a higher, semantic level as compared to the physical level of database schema (Grimm and Volz, 2007; Uschold and Gruninger, 2004). Thus, ontologies can be used to integrate disparate databases and knowledge sources.

We use the term ontology to mean a data model that is used to represent different concepts or entities in a domain of knowledge along with their relationships. Stated mathematically, we define an ontology $\mathcal{O}$ as quadruple $(\mathcal{C},\mathcal{I},\mathcal{P},\mathcal{R})$ where $\mathcal{C}$ is a set of classes or concepts in the domain of interest, $\mathcal{I}$ is a set of instances or individuals of a class, $\mathcal{P}$ is a set of properties or roles of classes and $\mathcal{R}$ is a set of relations between classes, e.g. *isA*, *subClassOf*, *sameClass*. Within a Car domain, we can have a class MusicPlayer with an instance CDPlayer that has the property hasSixCDs. See Figure 4 for a simple example that shows part of a vehicular ontology with classes in boxes and attributes in ovals. It also shows the relationship between classes (*isA* and *hasA*).

Figure 4 Vehicle ontology (see online version for colours)



Knowledge representation languages with well-defined formal semantics are used to express ontologies so that they can be machine-processable. These include OWL (web ontology language) owl, Gellish (van Renssen, 2003), KIF (knowledge interchange format) kif and RIF (rule interchange format) rif, etc.

Description logics (DLs) are a family of knowledge representation languages that can denote a domain's knowledge in a structured and formal way. Important notions of the domain are represented by concept *descriptions* (Staab and Studer, 2004). That is, atomic concepts (unary predicates) and atomic roles (binary predicates) are used to form expressions using the concept and role constructors provided by the particular DL. They typically form decidable fragments of the first-order logic predicate calculus restricted to unary and binary predicates to capture the nodes and arcs in a network graph. The basic elements used to represent knowledge in the DL formalism are *concepts*, *individuals* and *roles*, where concepts are analogous to classes of things, individuals to instances and roles to relationships between things. DLs use concept constructors to build complex concepts out of simple ones.

The OWL DL and OWL Lite sub-languages of OWL can be translated into an expressive DL. Ontology entailment in OWL DL (respectively, OWL Lite) can be reduced to knowledge base (un)satisfiability in the DL $\mathcal{SHOIN}(\mathcal{D})$ (respectively, $\mathcal{SHIF}(\mathcal{D})$).

A DL knowledge base, or an ontology, contains statements about the domain of interest in the form of DL axioms and is composed of a *TBox* (terminological box) and an *ABox* (assertional box). The TBox captures the so-called terminological knowledge, which comprises general statements within a domain describing relationships between concepts, e.g. All men are mortal. TBox axioms can be thought of as class axioms. On the other hand, the ABox, captures the so-called assertional knowledge, which comprises statements about particular individuals and situations, e.g. Socrates is a man. ABox axioms describe relations between individuals and concepts.

Some standard reasoning tasks within the DL framework are:

- *Knowledge base satisfiability*: This is equivalent to validation of a knowledge base, also called *ontology consistency.*
- *Concept satisfiability*: To check if a given concept can have any instances, and is also called *ontology coherency.*
- *Instance checking*: This is a test for class membership by checking if an individual is an instance of a given concept. This task can be extended to *type inference* by determining all the named classes that serve as types for an individual.
- *Subsumption*: This is to check if a given concept is a subconcept of another one. This is considered a distinguishing feature of DLs as opposed to other formalisms, and is also known as *classification.*

We are mainly concerned with two reasoning tasks in our framework:

- *Validation*: To detect any contradictions in the modelled information in the ontology.
- *Deduction*: To derive implicit conclusions from the information in the ontology, capturing the notion of logical consequence.

2.4 System level verification and control

At design time, the manufacturer will populate its car ontology with the properties and roles of the various electronically capable elements of the car. This is done automatically as each element supplies its attributes to the ontological registry. Our implementation uses extensible markup language (XML) format for this information. In future, a standard ontology across cars and parts manufacturers will greatly facilitate this task. Information about non-electronic parts of the system can also be added. For example, a class OwnerFamily can be predefined with its specific values left to the actual owner to fill in through an interface.

An ontology provides a formal, machine-readable taxonomy of a system and brings the benefits of the semantic web (Berners et al., 2001) and its accompanying technologies. We have used the Protégé-OWL ontology editor (protege; Knublauch et al., 2004) to define our car ontology. The ontological model is stored internally in description logic (OWL DL) (owl, 2004) and provides reasoning and classification abilities. A reasoning engine can make explicit all the implicit relationships (e.g. *subClassOf)* and perform consistency checks etc. The DIG (DL implementation group) compliant reasoner Racer (renamed abox and concept expression reasoner) (racer, n.d.) is employed to classify the taxonomy and to find inconsistencies in it.

A further ability is to define rules in description logic both by the car manufacturer at design time and by the car owner. In our system, rules, are written in SWRL (semantic web rule language) (swrl; O'Connor et al., 2005) and we use Jess (Java expert system shell) (Jess; O'Connor et al., 2005) to reason about them. SWRL is a syntactic and semantic extension of OWL DL. It is based on a combination of the OWL DL and OWL Lite sub-languages of the OWL Web Ontology Language with the Unary/Binary Datalog RuleML sub-languages of the rule markup language (RuleML). SWRL is a syntactic and semantic extension of OWL DL: it extends OWL axioms to include Horn-like rules. A *Horn clause* is a disjunction of literals with at most one positive literal.

Following is an example rule, in pseudo language, inserted at design time by the manufacturer:

$$\text{if ((car key in ignition)} \wedge \text{(door open))} \rightarrow \text{(sound alarm)} \tag{1}$$

Following is an example rule, added by the car owner:

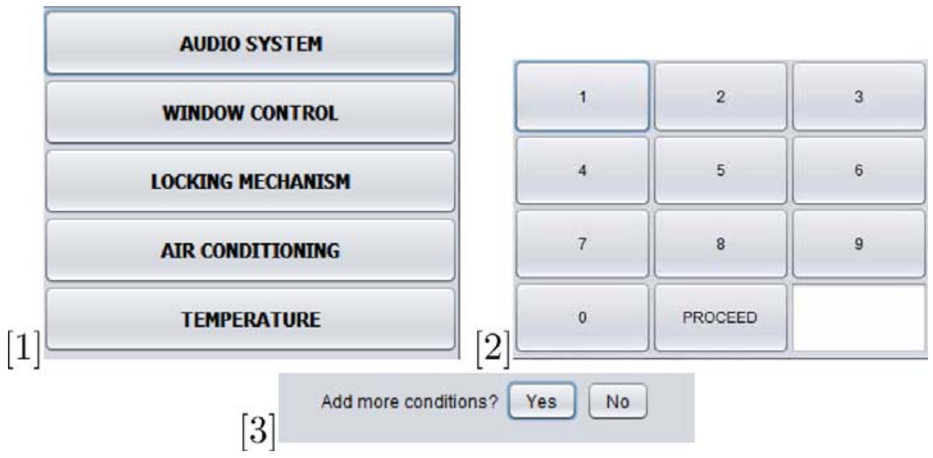
$$\text{if ((number of passengers)} > 1) \rightarrow \text{(lock doors when car starts)} \tag{2}$$

Rules give the user means for regulating the car network's behaviour, appropriate for many forms of specification. Rule engines vary in their performance and memory use but Protégé provides flexibility by allowing selection of the best rule engine for a given domain.

SWRL rules are stored as OWL individuals with their associated knowledge base. All the OWL instances and SWRL rules are automatically mapped onto Jess facts and Jess rules, respectively. Any new classifications or properties resulting from executing the Jess inference engine are written back to the OWL knowledge base. Although SWRL is an undecidable superset of OWL DL, it can be restricted to assure decidability.

To simplify rule writing, we have designed a Java application which provides the user with a simple graphical user interface (GUI) using buttons to input rules. Let us consider a rule example where the user wants to automatically turn on the air-condition (AC) if the temperature is over 30°. From the main menu, the user presses the 'Temperature' button and is prompted to enter the temperature value. She enters 30. The next menu presents a list of components that can be controlled with temperature, which in this case are Windows and AC. If AC is selected, another prompt asks whether the AC should be turned ON or OFF at a temperature of 30°. The user selects ON.

Figure 5 GUI screenshots



Some screenshots of this process are given in Figure 5. At the back end, the following SWRL rule is generated:

$$\text{Car}(?c) \wedge \text{hasTemperature}(?c, 30) \rightarrow \text{isACOn}(?c, \text{true}) \tag{3}$$

Table 1 shows the XML automatically generated for this rule. This application can easily be ported to a car where it can be accessed via a touch screen on the dashboard/console.

An important question is where to store the ontology? As the ontology serves as a registry for all the components, it is not feasible or useful to store all of it on-board the vehicle. hi fact, there is a strong case for not allowing the owners of the car access to the fine granularity component level tasks for their own safety and for proper functioning of the system. It suffices to save only those non-safety critical parts of the ontology within the vehicle that can be used in logic rules for passenger comfort. Some of these rules will be predefined at the factory and others will be defined by the owners, as discussed above. However, the rule base stored in the vehicle will contain all the relevant safety rules inserted at design time by the manufacturer to prevent any user-defined ones from violating them. For example, an owner might define a rule that involves opening the boot of the car while the car is moving. This would be caught by the reasoner as being in conflict with a preloaded unalterable safety rule that prevents the boot of a moving car from opening.

Other systems have been proposed that make use of ontologies, e.g. vehicle assembly plant diagnostics system (Chougule and Chakrabarty, 2009) and dynamic path planning of autonomous vehicles (Provine et al., 2004). Angele et al. (2008) give a semantics-based approach for improving the process of testing different configurations of cars.

3 Feature interaction check at design time

In this section, we describe the AUTOSAR ECU configuration process and illustrate how we have extended it to check for feature interaction.

Table 1 ACOn.xml XML produced for rule 3

```
<swrl:Imp rdf:about="http://www.owl-ontologies.com/ACRule">
   <swrl:body>
     <swrl:AtomList>
       <rdf:rest>
         <swrl:AtomList>
           <rdf:first>
             <swrl:DatavaluedPropertyAtom>
               <swrl:propertyPredicate rdf:resource="#
                   hasTemperature"/>
               <swrl:argument2 rdf:datatype="http://www.w3.org
                   /2001/XMLSchema#long"
               >30</swrl:argument2>
               <swrl:argument1>
                 <swrl:Variable rdf:ID="c"/>
               </swrl:argument1>
             </swrl:DatavaluedPropertyAtom>
           </rdf:first>
           <rdf:rest rdf:resource="http://www.w3.org
               /1999/02/22-rdf-syntax-ns#nil"/>
         </swrl:AtomList>
       </rdf:rest>
       <rdf:first>
         <swrl:ClassAtom>
           <swrl:argument1 rdf:resource="#c"/>
           <swrl:classPredicate rdf:resource="#Car"/>
         </swrl:ClassAtom>
       </rdf:first>
     </swrl:AtomList>
   </swrl:body>
   <swrl:head>
     <swrl:AtomList>
       <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf
           -syntax-ns#nil"/>
       <rdf:first>
         <swrl:DatavaluedPropertyAtom>
           <swrl:propertyPredicate rdf:resource="#isACOn"/>
           <swrl:argument1 rdf:resource="#c"/>
           <swrl:argument2 rdf:datatype="http://www.w3.org
               /2001/XMLSchema#boolean"
           >true</swrl:argument2>
         </swrl:DatavaluedPropertyAtom>
       </rdf:first>
     </swrl:AtomList>
   </swrl:head>
 </swrl:Imp>
```

AUTOSAR concepts are defined in the form of a domain specific XML exchange format (Pagel and Br, 2006). This format enables a seamless configuration process of the basic software stack and integration of application software in ECUs. The ECU configuration process at design time consists of four steps:

1 configure system

2 extract ECU-specific information

3 configure ECU

4 generate executable to burn/flash the ECU.

To check for feature interaction, we have added an extra step (shown in Figure 6) at the beginning of the standard AUTOSAR configuration process. In this step, the ontology of different ECUs with SWRL rules is provided as input to Protégé. If any feature interaction is detected, a flag is raised and the configuration process terminates.

In the automotive domain, a seminal example is the feature interaction between safety and security features (Calder et al., 2003; Juarez-Dominguez et al., 2007). The job of the security feature is to ensure that the car is locked at all times while the safety feature ensures the well-being of the occupants. In case of an accident or a crash, the safety mechanism unlocks all doors automatically to allow the passengers to get out of harm's way. But this feature also exposes a security pitfall: if a thief hits the front of a stationary to mimic a crash, all doors will automatically unlock, which is in conflict with the job of the security feature. On the component level, this interaction takes place between the Crash ECU and the Theft Deterrent ECU. The ECU common to both of these is the Door Lock ECU. In case of a pseudo crash, the Crash ECU tells the Door Lock ECU to unlock the doors. This is contrary to the role of the Theft Deterrent ECU.

To catch such kind of feature interaction, our enhancement to the AUTOSAR pipeline makes use of ontology and rule bundles. The ontology follows the structure and connections of the ECU configuration XML and the SWRL rules depict the behaviour of the ECU software. We envision that in future, car manufactures will provide a rule bundle with every ECU. These rules would define the outlook of the system in various scenarios. Security and safety ontologies are shown separately in Figures 7 and 8. When both of these features in the form of ontologies are mapped in Protégé at design time, it would raise a feature interaction flag after running them through an inference engine, depicting that the current rule bundles are inconsistent.

The security bundle consists of the following rules:

$$\begin{aligned}&\text{RemoteLockControl}(?r) \wedge \text{isOpenPressed}(?r, \text{true})\\ &\qquad \wedge \text{DoorLock}(?d) \rightarrow \text{isLocked}(?d, \text{false})\end{aligned} \tag{4}$$

$$\begin{aligned}&\text{RemoteLockControl}(?r) \wedge \text{isClosePressed}(?r, \text{true})\\ &\qquad \wedge \text{DoorLock}(?d) \rightarrow \text{isLocked}(?d, \text{true})\end{aligned} \tag{5}$$

Figure 6 Enhanced AUTOSAR ECU configuration (see online version for colours)

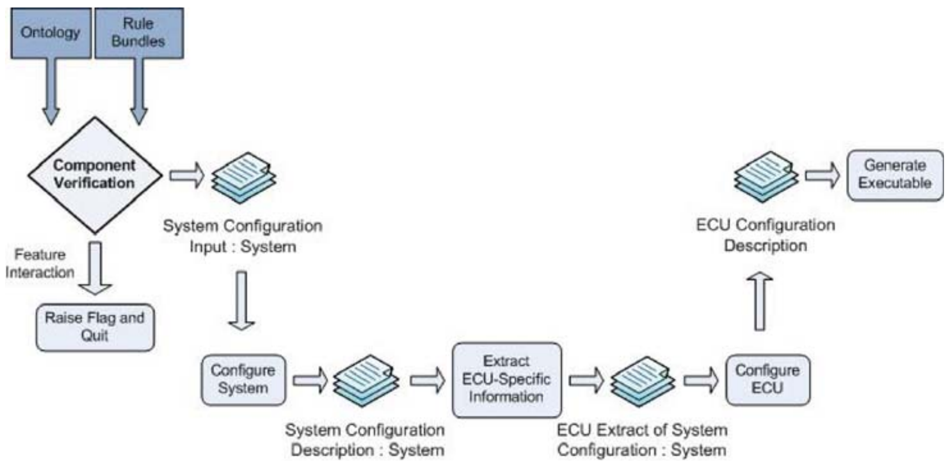

Ontology
Rule Bundles
Component Verification
Feature Interaction
Raise Flag and Quit
System Configuration Input : System
Configure System
System Configuration Description : System
Extract ECU-Specific Information
ECU Extract of System Configuration : System
Configure ECU
ECU Configuration Description
Generate Executable

Figure 7 Security ontology (see online version for colours)

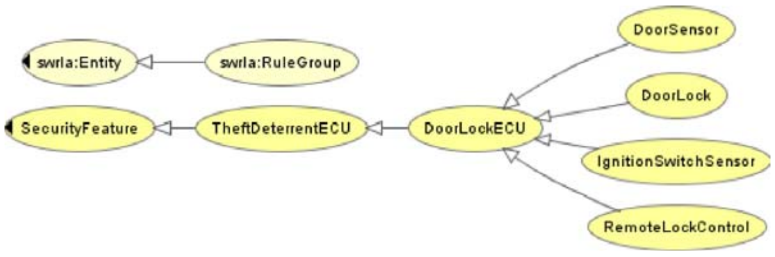
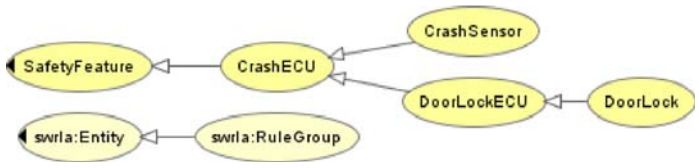


Figure 8 Safety ontology (see online version for colours)



Rule (4) unlocks all doors when the lock button on the remote control is pressed while Rule (5) locks them.

$$\begin{aligned}&\text{IgnitionSwitchSensor(?i)} \wedge \text{keyInIgnition(?i, true)}\\ &\quad \wedge \text{DoorSensor(?c)} \wedge \text{isClosed(?c, true)}\\ &\quad \wedge \text{DoorLock(?d)} \rightarrow \text{isLocked(?d, true)}\end{aligned} \tag{6}$$

The job of Rule (6) is to lock all doors when the key is in the ignition and all doors are closed.

$$\begin{aligned}&\text{IgnitionSwitchSensor(?i)} \wedge \text{keyInIgnition(?i, false)}\\ &\quad \wedge \text{DoorSensor(?c)} \wedge \text{isClosed(?c, true)}\\ &\quad \wedge \text{RemoteLockControl(?r)}\\ &\quad \wedge \text{isOpenPressed(?r, false)}\\ &\quad \wedge \text{DoorLock(?d)} \rightarrow \text{isLocked(?d, true)}\end{aligned} \tag{7}$$

Rule (7) ensures that all doors remain closed when the car is parked.

The safety bundle primarily consists of the following rule:

$$\begin{aligned}&\text{CrashSensor(?cs)} \wedge \text{isCrash(?cs, true)}\\ &\quad \wedge \text{DoorLock(?d)} \rightarrow \text{isLocked(?d, false)}\end{aligned} \tag{8}$$

Rule (8) force unlocks all doors when there is a crash.

When both of these ontologies and rule bundles are combined, any feature interaction between the two can be detected. In case of a pseudo crash, there would be a conflict between Rules (7) and (8) with the former trying to keep the doors locked, while the latter force-unlocking them. At design time, as soon as this conflict is detected, the ECU configuration mechanism will come to a halt and raise a flag due to functional nature of the datatype property *isLocked*, suggesting the current rule bundle is inadequate to define the overall behaviour of the components. To resolve this conflicts the car manufacturer would have to modify Rules (7) and (8) to Rule (9) and Rule (10) with the resulting ontology shown in Figure 9.

$$\begin{aligned}\text{IgnitionSwitchSensor(?i)} &\wedge \text{keyInIgnition(?i, false)}\\ &\wedge \text{DoorSensor(?c)} \wedge \text{isClosed(?c, true)}\\ &\wedge \text{RemoteLockControl(?r)}\\ &\wedge \text{isOpenPressed(?r, false)}\\ &\wedge \text{PassengerOccupancySensor(?p)}\\ &\wedge \text{passengersPresent(?p, false)}\\ &\wedge \text{DoorLock(?d)} \rightarrow \text{isLocked(?d, true)} \qquad (9)\end{aligned}$$

$$\begin{aligned}\text{CrashSensor(?cs)} &\wedge \text{isCrash(?cs, true)}\\ &\wedge \text{PassengerOccupancySensor(?p)}\\ &\wedge \text{passengersPresent(?p, true)}\\ &\wedge \text{DoorLock(?d)} \rightarrow \text{isLocked(?d, false)} \qquad (10)\end{aligned}$$

Rule (10) will ensure that the doors should be unlocked only if there are passengers in the car, thus complying with the goals of the security feature.

We have also devised a method to handle feature interaction in a more intelligent way: by prioritising applications. As a result, pulling down of a car window by a passenger can take precedence over the AC system's goal of a lower temperature on a hot summer day. Or, a proximity sensor output can make the system take control of the car and may limit manoeuvring of the car by the driver. This is described in more detail in the next section.

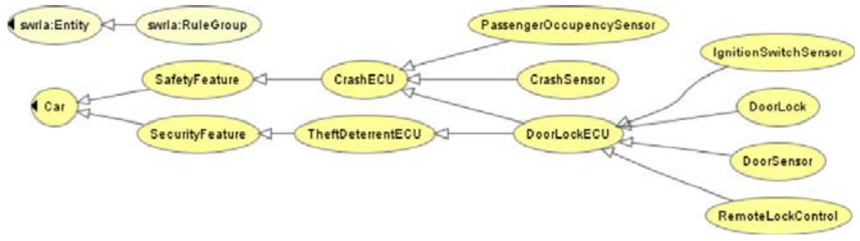
3.1 CAN-bus prioritisation

In this section, we introduce controller area network (CAN) and describe how to make use of it to filter out unwanted packets sent by compromised ECUs.

CAN is a high-speed serial network to connect microcontrollers and devices within an automotive environment (Tong et al., 2007). It employs a message broadcast-based protocol. Each entity within the network can send and receive messages, but not at the same time. Messages are transmitted serially onto the bus with priorities based on IDs. Some of its features include:

- fast reaction time
- multi-master communication
- high level of reliability
- low physical medium cost.

Figure 9 Combined safety and security ontology (see online version for colours)



Our testbed consists of a Linux machine which makes use of SocketCAN[3] to prioritise different features. SocketCAN is an open source implementation of CAN contributed to the Linux kernel by Volkswagen Research[4]. It can be configured to run in the standard Linux environment by enabling the CAN-bus module and recompiling the kernel. The SocketCAN concept extends berkeley software distribution (BSD) sockets by introducing a new protocol family *PF_CAN.* To test our feature prioritisation mechanism, we created a virtual network of ECUs which could transmit and receive data frames. The transmission and reception part of the code is shown in Tables 2 and 3, respectively. In Table 2, *frame.can_id* holds the CAN ID of the virtual ECU while *frame.data* holds the data frame to be transmitted. We also coded a filter capable of blocking frames on the basis of their CAN IDs. Frames from different ECUs were blocked at random assuming that they contributed to feature interaction.

Table 2 Sending a SocketCAN frame

```
struct can_frame frame;
frame.can_id =20;
strcpy( (&frame . data , argv [1] ) ;
frame.can_dlc = strlen ( &frame.data );
int bytes_sent = write( skt, &frame , sizeof(frame) );
```

Table 3 Receiving a SocketCAN frame

```
int bytes_read = read (skt, &frame, sizeof (frame) ) ;
priintf ("Read : \%s|n", &frame . data);
```

In the actual CAN-bus, this filter exists in hardware. Our Domain Manager will control the hardware filter by setting register values. For example, if pulling down of a window has a higher priority than the AC then, when they interact, the Domain Manager will tell the filter to drop frames sent by the AC ECU, based on its CAN ID. Hence, no feature interaction will take place.

Many other rules can be written to make use of CAN-bus prioritisation:

- any speed limits coming in real time from traffic management centre will have higher priority
- a crash (internal sensor) will have higher priority
- a message from the vehicular network informing about an accident somewhere on the road in front will have higher priority
- a message or a sensor input about decreasing speed of the front car will have higher priority.

4 Location based services

Location-based services (LBS) make use of contextual information of users and/ or devices to provide some service. According to Junglas and Watson (2008), in the four-year period from 2006 to 2010 the LBS market of Asia was expected to grow from $291.7 million to $447 million, Europe's from $191 million to $622 million and the US market from $150 million to $3.1 billion. Activity inference can also be activated as a result of location detection (Dey et al., 2010) which will provide long-lasting services or experiences. LBS researchers agree that there are two types of LBSs: location-tracking services in which parties/entities other than the user track the user's location and position-aware services in which the device makes use of its own location (Barkhuus and Dey, 2003). It is important to distinguish between these two because of the implications for privacy and information security.

LBSs also exist in the automotive domain, of which car navigation systems are one example. Our system also allows LBS to be handled by the Domain Manager (introduced in Section 2.2) which contains rules on how to process location information and which ECUs to forward it to. It has a list of GPS coordinates and information about the locations that these coordinates represent. We illustrate this with an example.

Traditional cruise control systems have a 'speed limiter' function with a pre-set maximum speed limit to keep the acceleration of the car in check. Our system enables this maximum speed limit to be dynamic depending on the location. The Domain Manager has a bundle of speed limits. It receives GPS coordinates from the GPS/ navigation system ECU (through the CAN-bus) and maps these onto speed limits and forwards them to the cruise control ECU (again over the CAN-bus). For example, let the driver get on to Road X. From the coordinates received from the GPS ECU, the Domain Manager will know that Road X has started. It will match this information with the speed limits list and forward a limit, say 60 km hr^{-1}, to the cruise control ECU, which will in turn set it as its speed limit.

The SWRL rules are:

$$\begin{aligned}&\text{Car}(?c) \wedge \text{hasGPSXCoordinate}(?c, 20.01)\\&\qquad \wedge \text{hasGPSYCoordinate}(?c, 30.02)\\&\qquad \rightarrow \text{hasLocation}(?c, \text{RoadX})\end{aligned} \tag{11}$$

$$\begin{aligned}&\text{Car}(?c) \wedge \text{hasLocation}(?c, \text{RoadX})\\&\qquad \rightarrow \text{hasCruiseLimit}(?c, 60)\end{aligned} \tag{12}$$

The speed limits will be provided by the local traffic law enforcement.

5 Advantages of the two-pronged approach

In general, an ontology is used as a domain model to share, reuse, analyse and better engineer knowledge (Jones et al., 1998). A particularly useful benefit is the separation of domain knowledge from operational knowledge (Noy and Mcguinness, 2001), which enables the development of applications without requiring specialised knowledge of a particular domain. Hence, an ontological model serves as a convenient data reflection API for programmers while avoiding the rigid schema of databases. This allows systems to evolve better.

Combining checkable application digests with rules in the DL-based ontology server allows heterogeneous components to interact correctly and reliably. This means that semantic interoperability (the unambiguous and correct interpretation of exchanged data by two systems), safety and reliability are ensured in the composed system. This approach has the following advantages:

- conflicting behaviour, or feature interaction, is avoided in the car network domain of interacting components
- overall system behaviour can be constrained by a set of overseeing rules.

Adequate tool support is available, both off-the-shelf and developed by us, to enable checkable and reliable real-time control of the pervasive system comprising heterogeneous components. The system is extensible because the ontological representations of new parts of the knowledge base can be added in a consistent manner.

In our architecture, the Domain Manager can also leverage contextual information to provide various services. As shown in the previous section, these services can be both safety and leisure related.

6 Related work

Cilia and Buchmann (2002) make use of web services and active databases for customising user behaviour in a vehicle. Data and metadata are interpreted by using ontologies as shared concepts. An ontology service is used to expose, store and manage all the concepts defined in the ontologies. Eigner and Lutz (2008) present a system to avoid vehicular collisions and accidents by using remote sensor data such as rain intensity, distance to the car in front, etc. This data exchange is made possible by VANETs. Contextual information is defined in the form of ontologies which can also be used by other applications. Sun et al. (2010) have also employed ontologies to represent contextual information. This contextual information is used to model the relationship between different components inside a smart car. A conversational dialog system for automobile drivers is described by Yan et al. (2007). The system enables the driver to control different components of the car such as the stereo and navigation systems. All domain knowledge is structured in the form of an ontology. The only other work that we know of that comes close to our work has been presented by Sandkuhl and Billig (2007) in which ontologies are used for integrated management of artefacts in automotive electronics. Such artefacts include metamodels, documents, metadata, etc. Although, this method enables car manufacturers to manage a wide array of product variations, it does not check for conflicts or feature interaction.

7 Conclusion and future work

We have shown that employing an ontological control plane with description, logic semantics is conducive to integration with a rule-based approach to network control. We have adapted our previous work on smart homes to develop a two-tier verification and control architecture for the car domain. We use open source software tools and the performance of the system is reasonable because the intensive reasoning part is predominantly performed at design time. Only computationally minor reasoning tasks are performed during actual use of the vehicle. Through an enhancement to the AUTOSAR standard, we have shown that feature interaction can be mitigated at the design stage. Moreover, our architecture can make use of rule bundles and location information to provide a wide variety of services to vehicles.

Our future work includes running quantitative and qualitative tests on the useability of our system. In this context, we also plan to leverage our earlier work in LBS (Ayres et al., 2009) and distributed virtualisation (Mehmood et al., 2005). We are developing a comprehensive ontology for the car domain that can serve as a starting point in the future standardisation efforts. We are also working on improving the efficiency of automated reasoning in our system. Moreover, we are exploring different methodologies to protect on-board components from external attacks. To this effect, we are in the process of implementing a novel inter-VANET firewall.

References

Road vehicles interchange of digital information controller area network (can) for high speed communication. ISO/DIS 11898, February 1992.

Albus, J. et al. (2002) *4D/RCS Version 2.0: A Reference Model Architecture for Unmanned Vehicle Systems*, Gaithersburg, MD: National Institute of Standards and Technology. Available online at: http://www.isd.mel.nist.gov/projects/rcs/ref_model/coverPage3.htm.

Alvi, A. and Greaves, D.J. (2006) 'A logical approach to home automation', *IE'06: Proceedings of the Second IET International Conference on Intelligent Environments*, IET, ISBN 0-86341-663-2, Vol. 2, pp.45–50.

Alvi, A. and Greaves, D.J. (2008) 'Checkable domain management with ontology and rules', in *Proceedings of the 2008 Third International Conference on Internet and Web Applications and Services (ICIW'08)*, IEEE Computer Society, Washington, DC, USA, pp.142–149. DOI: 10.1109/ICIW.2008.65. Available online at: http://dx.doi.org/10.1109/ICIW.2008.65.

Angele, J., Erdmann, M. and Wenke, D. (2008) 'Ontology-based knowledge management in automotive engineering scenarios', *Ontology Management Semantic Web, Semantic Web Services, and Business Applications*, Vol. 7, pp.245–264.

AUTOSAR (Automotive Open System Architecture). Available online at: http://www.autosar.org.

Ayres, G., Mehmood, R., Mitchell, K. and Race, N.J.P. (2009) 'Localization to enhance security and services in wi-fi networks under privacy constraints', *First International ICST Conference, EuropeComm'2009*, ISBN 978-3-642-11283-6, pp.175–188.

Barkhuus, L. and Dey, A. (2003) 'Location-based services for mobile telephony: a study of users' privacy concerns', in *Proceedings of the 9th IFIP TC13 International Conference on Human-Computer interaction (INTERACT'2003)*, pp.709–712.

Berners-Lee, T., Hendler, J. and Lassila, O. (2001) 'The semantic web', *Scientific American*, Vol. 284, No. 5, pp.34–43.

Berwanger, J., Ebner, C., Schedl, A., Belschner, R., Fluhrer, S., Lohrmann, P., Fuchs, E., Millinger, D., Sprachmann, M., Bogenberger, F., Hay, G., Krger, A., Rausch, M., Budde, W.O., Fuhrmann, P. and Mores, R. (2001) 'Flexray – the communication system for advanced automotive control systems', *SAE 2001 World Congress*, Detroit, MI, USA, March 2001, Session: Safety-Critical Systems, Paper No.: 2001-01-0676, DOI: 10.4271/2001-01-0676.

Bowen, T.F., Dworack, F.S., Chow, C.H., Griffeth, N., Herman, G.E. and Lin, Y-J. (1989) 'The feature interaction problem in telecommunications systems', *Proceedings of the Seventh International Conference on Software Engineering for Telecommunication Switching Systems (SETSS'89)*, pp.59–62.

Burns, A. (1999) 'The ravenscar profile', *Ada Letters*, Vol. 12, No. 4, pp.49–52.

Calder, M. and Magill, E.H. (2000) *A Conceptual Basis for Feature Engineering*, Amsterdam, The Netherlands: IOS Press.

Calder, M., Kolberg, M., Magill, E.H. and Reiff-Marganiec, S. (2003) 'Feature interaction: a critical review and considered forecast', *Computer Networks*, ISSN 1389-1286, DOI: http://dx.doi.org/10.1016/S1389-1286(02)00352-3, Vol. 41, No. 1, pp.115–141.

Cameron, E.J., Griffeth, N.D., Lin, Y-J., Nilson, M.E., Schnure, W.K. and Velthuijsen, H. (1994) 'A feature interaction benchmark for in and beyond', *IEEE Communications Magazine*, pp.1–23.

Chougule, R. and Chakrabarty, S. (2009) 'Application of ontology guided search for improved equipment diagnosis in a vehicle assembly plant', *Proceedings of the Fifth Annual IEEE International Conference on Automation Science and Engineering*, pp.90–95.

Cilia, M. and Buchmann, A.P. (2002) 'Profiling and service delivery in internet-enabled cars', *IEEE Data Eng. Bull.*, Vol. 25, No. 4, pp.66–70.

Dey, A., Hightower, J., de Lara, E. and Davies, N. (2010) 'Location-based services', *IEEE Pervasive Computing*, ISSN 1536-1268. DOI: http://doi.ieeecomputersociety.org/10.1109/MPRV.2010.10, Vol. 9, pp.11–12.

Edwards, W.K. and Grinter, R.E. (2001) 'At home with ubiquitous computing: Seven challenges', *UbiComp'01: Proceedings of the Third International Conference on Ubiquitous Computing*, London, UK: Springer-Verlag, ISBN 3-540-42614-0, pp.256–272.

Eigner, R. and Lutz, G. (2008) 'Collision avoidance in vanets – an application for ontological context models', *PERCOM'08: Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*, Washington, DC, USA: IEEE Computer Society, ISBN 978-0-7695-3113-7, DOI: http://dx.doi.org/10.1109/PERCOM.2008.42, pp.412–416.

Greaves, D.J., Gordon, D., Alvi, A. and Omitola, T. (2008) 'Using a .net checkability profile to limit interactions between embedded controllers', *Proceedings of the 2008 Second International Conference on Sensor Technologies and Applications (SENSORCOMM'08)*, Cap Esterel, France: IEEE Computer Society, pp.555–561.

Grimm, S. and Volz, R. (2007) *Semantics at Work, Ontology Management Tools and Techniques*, Chapter Foundations, eBook. Available online at: http://www.lulu.com/content/1969742.

Hsu, R.C. and Chen, L.R. (2005) 'An integrated embedded system architecture for invehicle telematics and infotainment system', *Industrial Electronics Proceedings of the IEEE International Symposium*, Vol. 4, pp.409–1414.

jess. Java expert system shell (Jess). Available online at: http://herzberg.ca.sandia.gov/.

Jones, D., Bench-Capon, T. and Visser, P. (1998) 'Methodologies for ontology development', *Proceedings of the IT&KNOWS Conference, XVIFIP World Computer Congress.* Available online at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.52.2437&rep=rep1&type=pdf.

Juarez-Dominguez, A.L., Joyce, J.J. and Debouk, R. (2007) 'Feature interaction as a source of risk in complex software-intensive systems', *25th International System Safety Conference.*

Junglas, I.A. and Watson, R.T. (2008) 'Location-based services', *Communications of the ACM*, ISSN 0001-0782, DOI: http://doi.acm.org/10.1145/1325555.1325568, Vol. 51, No. 3, pp.65–69.

Keck, D.O. and Kuehn, P.J. (1998) 'The feature and service interaction problem in telecommunications systems: a survey', *IEEE Transaction on Software Engineering*, Vol. 24, No. 10, pp.779–796.

kif. Knowledge Interchange Format (KIF). Available online at: http://www.ksl.stanford.edu/knowledge-sharing/kif/.

Knublauch, H., Fergerson, R.W., Noy, N.F. and Musen, M.A. (2004) 'The protégé owl plugin: an open development environment for semantic web applications', *International Semantic Web Conference – ISWC2004*, Vol. 3298, pp.229–243, DOI: 10.1007/978-3-540-30475-3_17.

Liu, L. and Özsu, M.T. (2009) *Encyclopedia of Database Systems*, New York: Springer-Verlag.

Mehmood, R. (2004) 'Disk-based techniques for efficient solution of large Markov chains', PhD Thesis, University of Birmingham.

Mehmood, R., Crowcroft, J., Hand, S. and Smith, S. (2005) 'Grid-level computing needs pervasive debugging', *GRID'05: Proceedings of the Sixth IEEE/ACM International Workshop on Grid Computing*, pp.186–193, Washington, DC, USA: IEEE Computer Society, ISBN 0-7803-9492-5, DOI: http://dx.doi.org/10.1109/GRID.2005.1542741.

Mehmood, R. and Crowcroft, J. (2005) 'Parallel iterative solution method for large sparse linear equation systems', *Technical Report*, UCAM-CL-TR-650, Computer Laboratory, University of Cambridge, UK.

Noy, N.F. and Mcguinness, D.L. (2001) 'Ontology development 101: a guide to creating your f irst ontology', Technical Report KSL-01-05, Stanford Knowledge Systems Laboratory. Available online at: http://www-ksl.stanford.edu/people/dlm/papers/ontology-tutorial-noy-mcguinness.pdf.

O'Connor, M.J., Knublauch, H., Tu, S.W., Grosof, B.N., Dean, M., Grosso, W.E. and Musen, M.A. (2005) 'Supporting rule system interoperability on the semantic web with SWRL', *International Semantic Web Conference*, pp.974–986.

owl. (2004) 'OWL web ontology language overview', *W3C recommendation*, 10 February 2004. Available online at: http://www.w3.org/TR/owl-features/.

Pagel, M. and Br, M. (2006) 'Definition and generation of data exchange formats in autosar', *Proceedings of the Second European Conference ECMDA-FA, MODEL DRIVEN ARCHITECTURE – FOUNDATIONS AND APPLICATIONS, Lecture Notes in Computer Science*, Vol. 4066/2006, pp.52–65, DOI: 10.1007/11787044_5.

protege. Protégé ontology editor. Available online at: http://protege.stanford.edu/.

Provine, R., Schlenoff, C., Balakirsky, S., Smith, S. and Uschold, M. (2004) 'Ontology-based methods for enhancing autonomous vehicle path planning', *Robotics and Autonomous Systems*, Vol. 49, pp.123–133.

racer. *RacerPro User's Guide*. Available online at: http://www.racer-systems.com/products/racerpro/users-guide-1-9-2-beta.pdf.

rif. (2006) 'RIF use cases and requirements', *W3C Working Draft*, 10 July 2006. Available online at: http://www.w3.org/TR/rif-ucr.

RuleML. *RuleML*. Available online at: http://www.ruleml.org/.

Sandkuhl, K. and Billig, A. (2007) 'Ontology-based artefact management in automotive electronics', *Int. J. Computer Integrated Manufacturing*, ISSN 0951-192X, DOI: http://dx.doi.org/10.1080/09511920701566467, Vol. 20, No. 7, pp.627–638.

Staab, S. and Studer, R. (2004) *Handbook on Ontologies (International Handbooks on Information Systems)*, Berlin, Heidelberg, Germany: Springer-Verlag.

Sun, J., Zhang, Y. and He, K. (2010) 'A petri-net based context representation in smart car environment', *Advances in Grid and Pervasive Computing, Lecture Notes in Computer Science, GPC*, Vol. 6104, pp.162–173.

swrl. (2004) 'SWRL: a semantic web rule language combining OWL and RuleML', *W3C Member Submission*, 21 May 2004. Available online at: http://www.w3.org/Submission/SWRL/.

Tong, W., Tong, C. and Liu, Y. (2007) 'A data engine for controller area network', *CIS'07: Proceedings of the 2007 International Conference on Computational Intelligence and Security*, Washington, DC: IEEE Computer Society, ISBN 0-7695-3072-9, DOI: http://dx.doi.org/10.1109/CIS.2007.15, pp.1015–1019.

Turner, C.R., Fuggetta, A., Lavazza, L. and Wolf, A.L. (1999) 'A conceptual basis for feature engineering', *Journal of System Software*, Vol. 49, No. 1, pp.3–15.

Uschold, M. and Gruninger, M. (2004) 'Ontologies and semantics for seamless connectivity', *SIGMOD Record*, ISSN 0163-5808, DOI: http://doi.acm.org/10.1145/1041410.1041420, Vol. 33, No. 4, pp.58–64.

van Renssen, A. (2003) 'Gellish: an information representation language, knowledge base and ontology', *Proceedings of the Third Conference on Standardization and Innovation in Information Technology*, pp.215–228, ISBN: 0-7803-8172-6, DOI: 10.1109/SIIT.2003.1251209.

Weiser, M. (1999) 'The computer for the 21st century', *SIGMOBILE Mobile Computer and Communication Review*, ISSN 1559-1662, DOI: http://doi.acm.org/10.1145/329124.329126, Vol. 3, No. 3, pp.3–11.

Yan, B., Weng, F., Feng, Z., Ratiu, F., Raya, M., Meng, Y., Varges, S., Purver, M., Lien, A., Scheideck, T., Raghunathan, B., Lin, F., Mishra, R., Lathrop, B., Zhang, Z., Bratt, H. and Peters, S. (2007) 'A conversational in-car dialog system', *NAACL'07: Proceedings of Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations on XX*, Morristown, NJ: Association for Computational Linguistics, pp.23–24.

Yousefi, S., Mousavi, M.S. and Fathy, M. (2006) 'Vehicular ad hoc networks (VANETs): challenges and perspectives', *ITS Telecommunications Proceedings, 2006 Sixth International Conference*, DOI: 10.1109/ITST.2006.289012, pp.761–766.

Notes

1 www.mono-project.com.

2 http://tomgruber.org/writing/ontology-definition-2007.htm.

3 http://developer.berlios.de/projects/socketcan/.

4 http://www.vwerl.com/.