

# **Smart Vehicular Traffic System**

***- Sagnik Bhattacharya***

There is no doubt that even today surface transport is the backbone of our transport infrastructure. The domain of a smart vehicular traffic system, can be subdivided into the following categories:

1. Smart Vehicles
2. Smart Roads
3. Interaction between the above two.

### **Visualization and Features :**

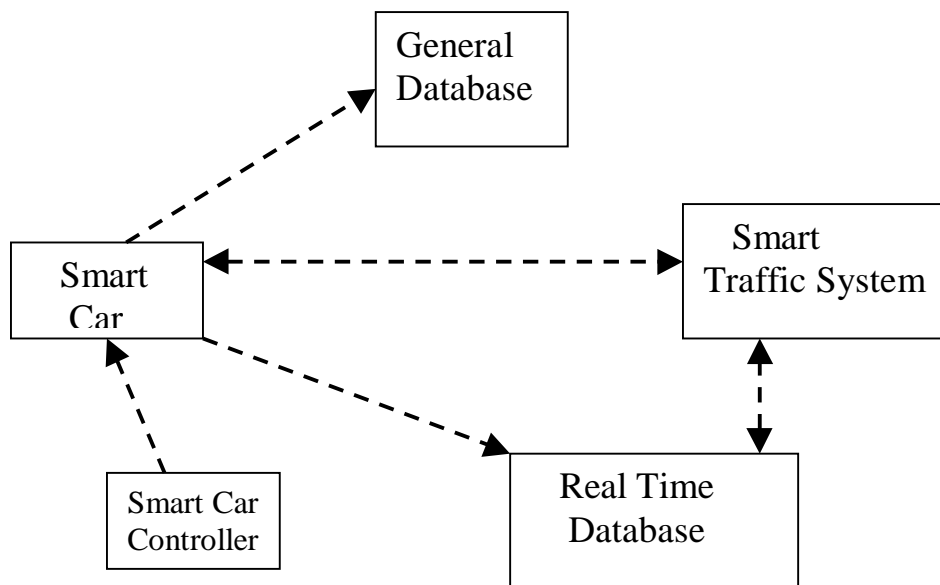
A person does not have to drive his/her own car. When the person approaches the car, the door is opened by voice activation. Then, it hears the name of the destination, and accesses some central database through some wireless communication protocol. The car then gets the best route available through some heuristic or algorithm, taking into account the traffic conditions, distance, traffic flows etc. As these data can change very rapidly, the best route can also vary with time. The car itself dynamically takes this decision. The destination can also be changed at any time by the passenger(!). The car, of course shall have some kind of vision/sensors to detect obstacles etc. It will also head for the nearest gas/electricity/service station, whenever the need arises. Some degree of fault tolerance shall be built into it. Each car shall have a comprehensive communication console built into it, which shall have functionalities of video conferencing, fax and news agents other than regular cellular phone access, and AM/FM access. Some sort of mechanism shall be provided which shall allow the passenger to be notified immediately of the various news items of interest to the passenger as and when they break over the television or radio or internet. In short we shall have a complete communication kit for the person, which shall integrate all means of personal communication including, telephone, video conferencing, e-mail, instant messaging, fax etc. Also there shall be some mechanism by which new devices can be plugged into the system and used.

The roads will have sensors at continuous rows of sensors to keep track of each and every car, its whereabouts and statistics like speed, exhaust level etc. There shall be pressure sensors on the roads to detect if the vehicle has exceeded its allowed payload. Smart traffic signals shall operate on the basis of traffic conditions and not on a direction-

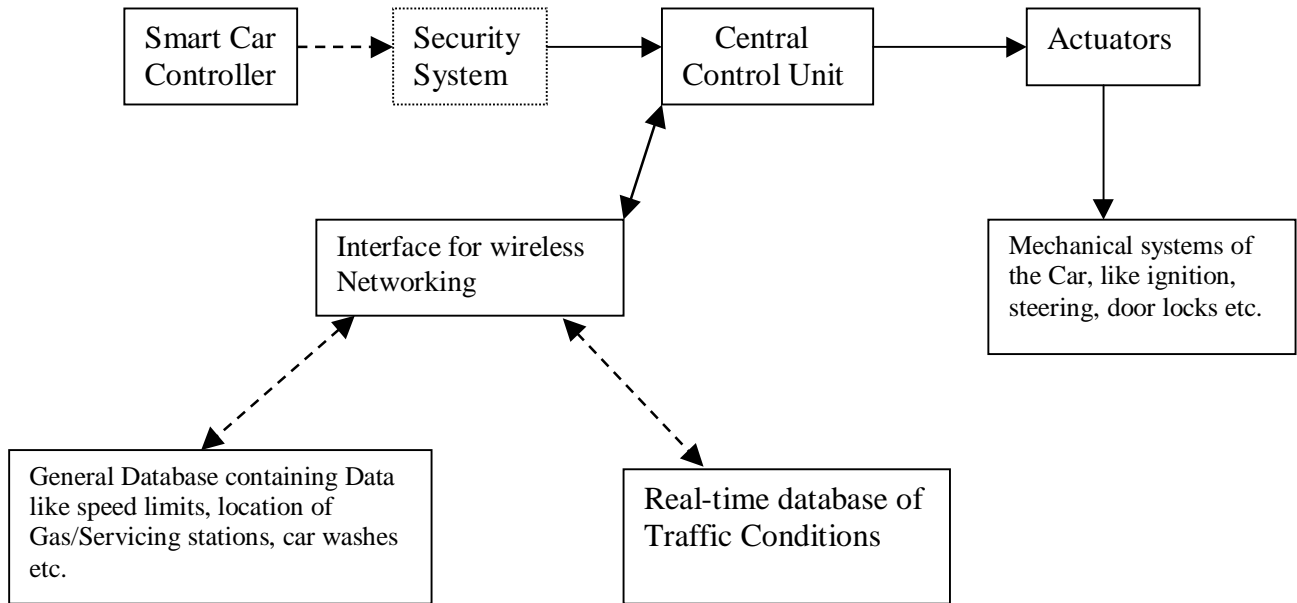
fair round-robin fashion. The cars shall get traffic signals automatically, so that no visualization is involved. Also, the car can get inputs like speed limits and road conditions from communication portals placed at periodic intervals. Some vehicles can have priorities greater than others, so they shall be supplied with fresher data than others and possible they shall have privileged access to certain data, which shall allow them to obtain some sort of a higher QOS differentiated service. For example, an ambulance which can break traffic lights on an emergency shall be given higher priority. On the other hand, a heavy duty truck, won't be allowed to be routed through residential areas.

Also, there shall be embedded web servers in the cars, so that if a person wants the car to come and pick him up, then he can just give the appropriate signal from his handheld device. Also the web browser can act as an easy to use interface for a mechanic to identify a problem if one arises.

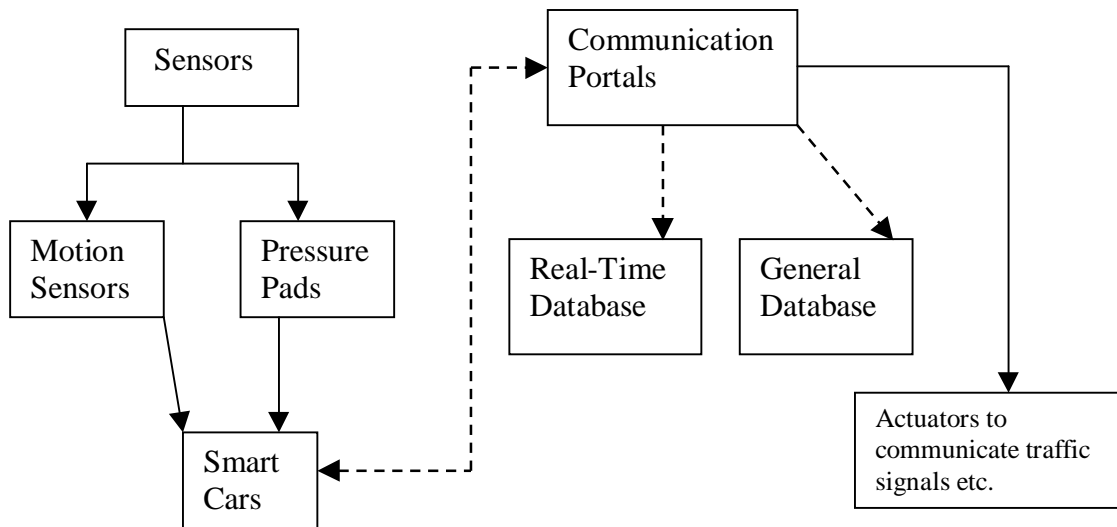
*A simplified block diagram of this smart space :*



***Block Diagram for the Smart Car :***



***Block Diagram for the Smart Traffic System :***



***Legend :***

A —————> B : A physically communicates with B.

A - - - - -> B : A communicates with B using wireless media.

## **Hardware Design :**

We can subdivide the hardware design problem into three parts : *design of the smart cars, design of the smart traffic systems and the design of the interfaces.*

### **Hardware design of the smart car :**

The smart controller is a small hand-held device, with an inbuilt DSP chip( which gets input from an ADC) for voice recognition. The DSP chip required has to consume low power, so a good choice would be C64X. Also, some secure memory needs to be provided, for which the memory stick can be used. The security system basically needs to validate the voice signature of the user and so an array/vector processor would do.

*The C64x has a wider address bus and a dual 17-bit MAC units. These features are especially important for wireless applications as they help in noise-reduction and extra precision in compression algorithms. Low power requirements are met by automatic power management and user-programmable “idle-domains”. Also the wider program bus and the new instruction buffer and primary cache help reduce bus activity, which saves a little more power.*

Now, the central control unit is the very heart of the smart car. It should have a very highly parallelisable RISC processor as it has to accomplish several critical tasks simultaneously. Power consumption is not a very big issue here, as the necessary power can be supplied from the car battery. It must be provide easy interfaces for various other devices and controllers. The determining factor in the choice of the processor would be the performance/cost ratio. Going by current trends, a P-3 or Alpha AXP 2 would do due to their high degree of parallelism, but logically speaking it is too early to decide on this as new and better technology shall be available before such a smart car can hit the market. The central control unit also drives a series of actuators which are essentially responsible for the functioning of the car. The type of actuators to be used include motors, valves, robot arms etc. There is also an interface for wireless communication with communication portals placed on the roads. These portals essentially provide a gateway for the communication between the car and the databases. The interface should have a highly parallel DSP processor of its own, like TigerSharc so as to effectively

communicate using some protocol like bluetooth(mainly due to its range), while not causing a bottleneck to the central control unit.

*TigerSHARC is particularly used as it has a register-based load-store architecture with a static superscalar dispatch mechanism in which instruction-level parallelism is determined prior to runtime under compiler or program control. It has a highly parallel, short-vector-oriented memory architecture with 128 fully interlocked registers.*

### **Hardware design for the Smart Traffic System :**

The smart traffic system has sensors placed at periodic intervals. These sensors include motion sensors to gauge the speed of the vehicles or to count the number of vehicles passing a point. The pressure pads detect payloads and can be used to ascertain if some vehicle is illegally parked etc. The communication portals which are placed at periodic intervals on the roads act as gateways for data access. They have a central DSP chip for elementary operations. But the major requirement is that it should be able to serve many clients at once. So, the DSP chip should support a high degree of multiprocessing. It needs to use some advanced communication protocols, because the range to the nearest database center maybe more than what a protocol like bluetooth might handle. Also that protocol should be able to provide very high data rates. These comm portals also act as small control unit to drive a series of actuators, which control the traffic lights etc.

### **Hardware design for the interface between the Smart Cars and the Smart Traffic System :**

The basic design issue here is the design of the databases. There are essentially two distributed databases. They are distributed so as to exploit geographical locality of

querying. The general database contains data like car registration information, speed limits, location of gas stations and other non-volatile data. The real-time database contains up-to-date information about the traffic conditions on various routes. Both these databases contain important information which might cause a problem if lost. Use of RAID reduces the chances of loss of data. Also, the real-time database needs very fast DMA, as the time-consistency of data is very low.

### **Network Design :**

As we can see from the block diagram of the smart space, there are two kinds of communications taking place between the components : wired and wireless. We consider the network design of the smart space first component-wise and then in terms of the interaction of the components.

### **Communication within the smart car :**

The smart controller has to communicate with the car by sending digitized voice signature, first for authentication and then for remote control. This is a strictly wireless communication, for which security is of prime importance. At the MAC level, bluetooth can be used because it provides for reliable communication over a range of 10 to 100 meters, but it certainly lacks the requisite security features. So, there is a need to put some kind of secure authentication protocol over bluetooth. This can be accomplished by putting some sort of PACS like local loop for the controller and the smart car.

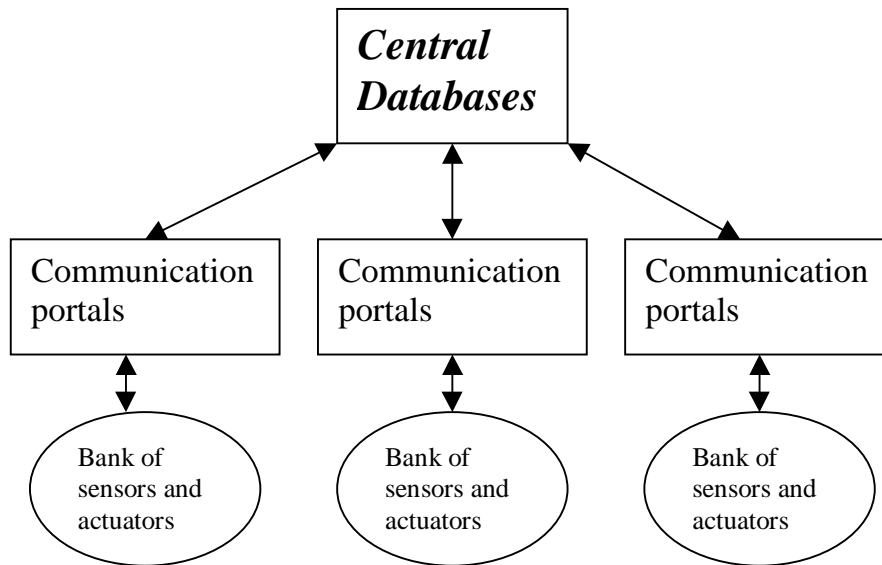
*The main motivation behind using Bluetooth is that instead of being a fixed protocol as such, it is a protocol stack, and any application can run over a particular vertical slice of the protocol stack. Thus we can design the higher level protocols such that they can efficiently interface with end-level applications (which might be proprietary or legacy applications) as long as it conforms with the specifications.*

*Also, Audio data can be transferred between one or more bluetooth devices, making various usage models possible and audio data in SCO packets is routed directly to and from Baseband. As the audio model is relatively simple within bluetooth, we use it to transfer voice data between the handheld car controller and the authentication unit/communication console of the car.*

As far as the rest of the car is concerned, the other modular communication can be carried out in wired mode. The voice activation of commands can be accomplished by sending the voice data to the executing unit over optical cables.

### **Communication within the smart traffic system :**

We can visualize our smart traffic system to be built in a tree like manner as shown below :



The comm portals communicate with their banks of sensors and actuators by forming a wireless local loop amongst themselves and PACS can be used to provide a data services at a low cost. At the lower level, the lower layers of the bluetooth protocol stack can be used, as it contains an inbuilt service discovery protocol (SDP). This provides for a lot of scalability in the system. At any point of time, new sensors and actuators can be added without having to reconfigure the whole system. The sensors pass various information like the speed, payload etc of a car and the number of car passing in

some time interval. The comm portals pass signals to the actuators that run the traffic signals. The reason for not using infra-red out here is that the comm portals might not be able to provide line of sight to the sensors and actuators. Also the Telephone control protocol provided by bluetooth shall support the car phone and video conferencing via the communication console.

*Discovery services are a crucial part of the whole framework. These services provide the basis for all the usage models. Using SDP, device information, services and the characteristics of the services can be queried and after that, a connection between two or more bluetooth devices can be established.*

*PACS supports wireline quality voice, voiceband data, digital data and messaging services and is based on a low-power, low-complexity design. Operating environments for PACS include indoor and outdoor; fixed, low mobility and vehicular mobility; and private and public access. PACS is a microcellular system providing high frequency reuse efficiency and thus is capable of supporting high traffic density. These attributes make PACS ideal for providing fixed wireless local loop in moderate to densely populated areas.*

The communication between the comm portals and the central databases is much more complex. As we had seen earlier, there are two kinds of distributed databases : a real-time database and a general database, both of whose components have to handle enormous amounts of traffic. So, obviously they'll need a very large buffer size to handle overload conditions in a failsafe way. A MAC protocol like a n-ack sliding window protocol with a window size of the order of 10000 would be appropriate. We cannot cellular-like communication because, the geographical locality is uncertain, even though it is fixed. Some kind of a WAP like protocol needs to be used. However it needs to have extended concurrency control features and should extend some QoS guarantees. There might be a need for multimedia data transfer. So, we need some kind of bandwidth-efficient, reliable multimedia transfer schemes. For this ITU-codecs can be used. Also service discovery protocol (SDP) needs to be supported, for scalability.

As far as the maintaining of consistency between the different parts of the distributed databases is concerned, it is done through high throughput B-ISDN band op-fibers network.

## **Communication between the smart cars and the smart traffic network :**

The modular design of the system shows that the car communicates with the traffic system only through the comm portals. As the car is a mobile entity, it is basically plugging into the local loop of a comm portal. So, it is basically acting as a mobile user. In concept this is similar to a cell phone network, but to have less overhead and to be able to communicate with more than one comm portal simultaneously. So, this calls for an implementation of the common access protocol (CAS), which is a kind of extension on the existing dual-function cell-phone AMPS/CDMA architecture. Some kind of multi-point access Iridium system could be used. Again some QOS guarantees are needed to ensure that the system does not crash due to excess traffic.

## **Important Issues :**

Security and reliability are of key importance in this system. We should be able to protect our system from malicious or fraudulent users. The features that are needed are :

- Clone prevention and detection services;
- Subscription fraud monitoring systems;
- Network security enhancement and monitoring systems;
- New mobile unit features that enhance fraud control, such as Authentication

## **Operating System Support :**

The operating system support needed for the smart vehicular traffic system can be divided into three basic types : firstly, support for the raw computational power needed by the central monitoring stations; secondly, component-based interactive support for the different parts of the traffic system and lastly, support for the embedded systems inside the smart cars.

Now the job of the central monitoring stations are very critical and hence require high real-time and fault-tolerant guarantees. Typically, each of the stations shall have a couple of hundred processors, with adequate main memory support, so that critical data can be retained in main memory, without having to page them back to disk. This is necessary for meeting the real-time requirements. One thing that could be done is to use commodity operating systems like IRIX etc. over a layer of Cellular Disco.

*A prime motivation for using Cellular Disco is that we can avoid hardware-partitioning and at the same time avoid the cost of developing newer operating systems. It uses the concept of virtual clusters on shared-memory multiprocessors. It takes care of the resource management, so that the commodity operating systems running on top of it is transparent to the whole Cellular Disco layer. However the most important feature of Cellular Disco which makes it so desirable is hardware fault-containment. If one node of the system fails, only the Virtual Machines running on those nodes shall terminate. The rest of the computation shall proceed normally. When the hardware has recovered, the system can function as previously.*

The component-based interactive-support for the traffic system can be provided by MMLite. The most compelling argument for this is that it is suitable for a wide variety of hardware and applications. What we need mostly in our comm portals is some kind of uniformity along with a certain amount of multithreaded capability. Also, we would like the components of our operating system to behave differently under different circumstances. For example, the scheduling policy could be different under heavy and light traffic conditions.

*The desirable aspect of the MMLite system architecture is that it supports real-time constraints irrespective of the type of microprocessor or network connection. Its components can be easily replaced and reimplemented. Also, it is efficient, portable and*

*has a very small memory footprint. MMLite components contain code and other metadata for classes of objects. When a component is loaded into an address space it is instantiated. This, in turn, creates object instances that communicate with other objects, potentially in other components. One very innovative aspect of MMLite is mutation. It is the act of automatically changing an ordinarily constant part of an object, such as a method implementation. For example, the system can switch from round-robin scheduling to FCFC scheduling automatically when it feels the need to do so.*

As far as providing the systems support for the smart car is concerned, we could use the KOALA component model for the non-essential car parts like the communication console. This is apt because the KOALA model was basically meant for consumer electronics components. Now, the smart car shall have hundreds of microcontrollers inside it, so for the system to integrate properly it should conform to some standard. Here's where the OSEK/VDX standard comes in. It is already widely accepted in European automotive circles. This standard comes in very handy as the cost and complexity of the microcontrollers keeps on increasing. Also, if at any point of time we want to introduce some new component or replace an old one with a better version, we just have to make sure that it conforms to the OSEK/VDX standard.

*The OSEK/VDX standard actually consists of three sub-standards : an operating system standard, a communication standard and a network manager standard. Though not a RTOS by itself, the integration of its various components gives it RT properties. For example, tasks can be basic or extended and preemptive or non-preemptive. It also has three levels of interrupt service routines and events are used to synchronize between the different tasks. Another important feature of the OSEK architecture is its error handling mechanism wherein it distinguishes between application and fatal errors. This way the microcontrollers can know when to take drastic measures.*

*The KOALA component model becomes important as the required diversity of products and their software is increasing rapidly. KOALA lets us apply the same software in different products, which saves product development effort. In KOALA, components are units of design, development and reuse. It communicates with its environment through interfaces. Though the components are designed independently, much of the interaction between them is taken care of by static binding. This is done to meet speed*

*constraints. It also allows dynamic binding at runtime. To support evolution of the various components, an interface repository and a component repository is maintained.*

### **Middleware and Software Support :**

This section is organized a little differently from the previous ones in the sense that here we look at various new technologies and try to see where they would be useful in our smart vehicular traffic system. Finally, a complete roundup of the arguments is presented.

### ***Real-Time Java :***

Java, as we all know, is becoming more and more popular every day, primarily because of its portability, and all the APIs it provides. Real-Time Java is one such package that, though in its nascent stage, aims to bring some real-time guarantees into the JVM. This would be helpful for software development for the comm portals and the central monitoring stations. We can use JDBC connectors on a real-time JVM to provide for the end-user support for our real-time databases.

*The issues that real-time Java addresses are : thread scheduling and synchronization, memory management, asynchronous event handling and physical memory access. Some sort of a real-time executive is used that takes care that spontaneous tasks are only accepted for execution only if they can be serviced without interfering with cyclic tasks scheduled previously. The RTSJ defines the Realtime Thread(RT) which the resident scheduler executes. RTs access objects on the heap and hence can incur delay from the garbage collector. So, the memory model is built such that the garbage collector does not interfere with the real-time execution of the threads.*

### ***CORBA, Jini, COIGN, HIVE :***

CORBA is definitely the middleware of my choice for the smart traffic system. Here we need a unifying base that hides the heterogeneity of the various components of the system. CORBA is based on the principle of interoperability. . Because of the easy

way that CORBA integrates machines from so many vendors, with sizes ranging from mainframes through minis and desktops to hand-helds and embedded systems, it is the middleware of choice for large (and even not-so-large) enterprises. The reason I would prefer it to Jini is that Jini can support only Java, so using it would mean that we cannot use much of the commodity software that are already available. With the same reasoning, COIGN shall be used to build distributed versions of the certain application, so as to provide some kind of software load balancing. Also, it would be preferable to HIVE, because, again, HIVE is Java-based. Though the decentralized architecture of HIVE is an appealing feature, it is not really required in this smart space.

*CORBA's architecture is based on Object Orientation, and built around three key building blocks: OMG Interface Definition Language (OMG IDL), the Object Request Broker (ORB) and the standard protocol (IIOP). CORBA applications are composed of objects. Typically, there are many instances of an object of a single type, all identical in functionality but differing in that each is assigned to a different customer, and contains data representing its particular customer. The interface to each object is defined very strictly. But, in contrast, the implementation of an object - its running code, and its data - is hidden from the rest of the system (that is, encapsulated) behind a boundary that the client may not cross. Clients access objects only through their advertised interface, invoking only those operations that that object chooses to expose, with only those parameters (input and output) that are included in the invocation. In this way CORBA ensures interoperability.*

*Jini technology is similar to CORBA to some extent. It provides a lookup service, a discovery/join protocol and a subtract-out mechanism called leasing. By using Java Remote Method Invocation, Jini infrastructure provides communication between objects across device boundaries that enables those objects to work together. RMI enables activation of objects and the use of multicast to contact replicated objects, providing high availability and high reliance objects to be easily implemented in the Jini framework. The Java virtual machine also protects the client machine from malicious downloaded code.*

*Coign is an Automatically Distributed Partitioning System (ADPS) for component-based applications. Given either a non-distributed or distributed application built from COM components, Coign will automatically create a new distributed version*

*of the application. Using scenario-based profiling, Coign creates an inter-component communication (ICC) model of the application. Later at execution time, Coign uses the ICC model to partition the application into client and server components and distribute the components across a network to reduce total communication costs. Coign operates on application binaries (.exe and .dll files); analysis, partitioning, and distribution are achieved with no access to application sources. Coign is language independent.*

*Hive consists of three components: cells, shadows, and agents. The Hive network is a decentralized collection of cells. A Hive cell is the analog to a web server, a program running on a specific computer with a published network address. Each cell contains a set of local resources called shadows that encapsulate capabilities such as a screen display or a digital camera. Each cell is also host to many agents that use local resources and communicate with each other. By analogy to a conventional operating system, a cell is like a kernel, shadows are like device drivers, and agents are like processes. The major difference between Jini and Hive is that Jini does not have the conceptual split between an agent and its shadow. Another important difference is Hive's location-dependent model. In Hive, an agent's cell is an important fact; it tells us where the agent is on the network, (potentially) where it is physically, what resources it has access to, etc. Jini focuses mostly on services; the actual place a service is hosted on is not a major part of the Jini model.*

So, now that we have our hardware requirements in place and we have placed our operating system requirements and specified our network management protocols, we are left with the decision of putting our endware in place. We discussed the topics of data fusion and embedded databases, and these are an integral part of our smart space. Data fusion helps the smart car to put together all the ambient information that it gets from its sensors and the comm portals and take decisions pertaining to the driving process. The smart car has a number of embedded databases in it which accomplish a number of functions. These databases store certain cached versions of the data that the car obtains from the comm portals, like the traffic congestion report for the next 5 miles, and they provide only limited functionality for querying, so as to have limited overhead and faster querying.

Voice XML, though in a very nascent stage of development, is a very promising technology. If the menu driven constructs of Voice XML could be modified so as to accommodate natural language processing, then we can simply embed a web server into the smart car controller and the user can interact with it using Voice XML.

### **Some other issues :**

Security of the system is of prime concern. We don't want any illegal access into the central databases, as they are critical to the whole system. In some ways CORBA is less secure than lets say Jini. So, we need to put a secure shell on top of the ORB to provide authentication. As of now , I am not elaborating much on the security aspects as we did not discuss any technical papers related to it.

Fault-tolerance is another important issue. Cellular Disco provides hard-ware fault-containment, but the software fault-containment is a problem that has to be addressed by the middleware/software layer. As for most parts, this is a hard real-time system, the software should be built keeping this in mind.

As, for evolution and reconfigurability, this is guaranteed by the use of standards and protocols, which are independent of commodity systems. For, example if we want to introduce smart motorcycles, all we need for it to gel with the system is that it conforms to the standards specifications.