Adapting the Auto to a New Tune

Johannes Helander
Microsoft Research
1 Microsoft Way
Redmond, WA 98052, USA
+1-425-882-8080
jvh@microsoft.com

Jürgo Preden
Tallinn University of Technology
Ehitajate tee 5
19086 Tallinn, Estonia
+372-620-2109
jurgo.preden@ttu.ee

ABSTRACT

Automobiles increasingly include a multitude of replaceable and configurable hardware and software components. The pieces need to interoperate with each other in real-time, with their environment, and with consumer computing devices. Future cars may be reconfigured on a daily basis, according to e.g. car rental agreements.

Current real-time systems are not well suited to highly dynamic and even chaotic environments. Analyzing the problem to death and assigning a fixed schedule may be feasible for closed systems such as the brakes, but cannot be done for every possible add-on combination and interaction with consumer devices.

This paper proposes a framework for adapting component interaction based on context histories and model based prediction. Tasks to be performed by a combination of software and hardware components are described in a pattern language, called the partiture. The partiture, analogous to the score of an orchestra, is used by the conductor to automatically configure, schedule, monitor, and adapt computation and communication of components.

Preliminary results demonstrate how a simple stochastic process can adapt a distributed entertainment scenario. This paper extends the concept to automotive applications with XML based interoperation, simple configuration, adaptive real-time, and security.

INTRODUCTION

Automobiles are increasingly configured to individual and changing customer needs due to competitive pressures. Meanwhile they are being built out of increasingly standardized and reusable components so as to make the manufacturing process effective. Embedded computers are being used to control, maintain, and monitor these components. It is clear that the software for those components has to be interoperable, adapt to changes in the environment, and customer desires.

We treat the car as a network of embedded computers, that not only provides interaction within the auto, but also between the vehicle itself and other consumer electronics, homes, car rental agencies (in rental cars), maintenance shops, and internet services. The critical parts of the vehicle are interacting with the real world and are thus inherently real-time systems. The entertainment systems are interacting with humans and are also real-time systems, as constrained by the resolution of the human senses.

Programming a highly adaptive distributed real-time system that comprises of changeable components and random consumer electronics devices is not well handled by traditional real-time methodologies. Traditional methods are more oriented towards a closed, well understood world that can be completely analyzed. Instead the relative chaos of the brave new world of interoperation and configurability requires adaptive approaches. Some previous work done in the area of run-time kernels deals with dynamically adapting computing loads, such as the work done in the context of the Spring project [1], using control theory on estimating simple repetitive tasks. However, no previous work has attempted to tune complex tasks spanning multiple nodes, base the adaptation on a declarative description, or use a stochastic process for real-time estimation. In addition our approach allows the definition of the configuration of an application (the involved nodes, the interaction patterns of nodes, etc) in a non-static way, allowing changes in the configuration after the system has been deployed and also introduction of new applications into an existing system without any change in system software.

The author shows in detail in [6] how to stochastically predict temporal requirements and adapt distributed schedules to real-time and reliability constraints. Even relatively simple stochastic methods appear promising, while utilizing technical analysis methods or fuzzy logic with the framework is likely to yield even better results.

Traditional programming methodologies are best suited for tightly coupled single computers. Programming a distributed process would be best done in a domain-specific language that is specifically designed for that purpose. This paper proposes such a language, called the *partiture* as it is analogous to the orchestra score that a conductor reads. It is a pattern that describes the required functions, message exchanges, timing requirements, and quality constraints. The architecture presented ties the partiture into a framework for scheduling, predicting, planning, monitoring, and executing distributed processes.

Given that tasks and processes that span multiple components in the automobile can be programmed and conducted, let us examine what kind of an orchestra the auto might turn out to be and what tunes it can waltz to.

Reconfiguring with Partitures

The goal is to make cars more flexible to allow reconfiguration of the car with minimum effort, even on the fly. The reconfiguration could apply to different aspects of the car itself – engine performance, transmission features, suspension features and various add-on features of the car – multimedia, security or navigation device. To allow for such change the architecture used to build these systems must allow for on-the-fly changes.

An entertainment example is playing music by streaming data from a recording device to speakers. Depending of the available number of speakers and their locations different partitures can be executed. If there are only two speakers, there is no way to play surround sound, which means that the partiture for surround sound can't be executed.

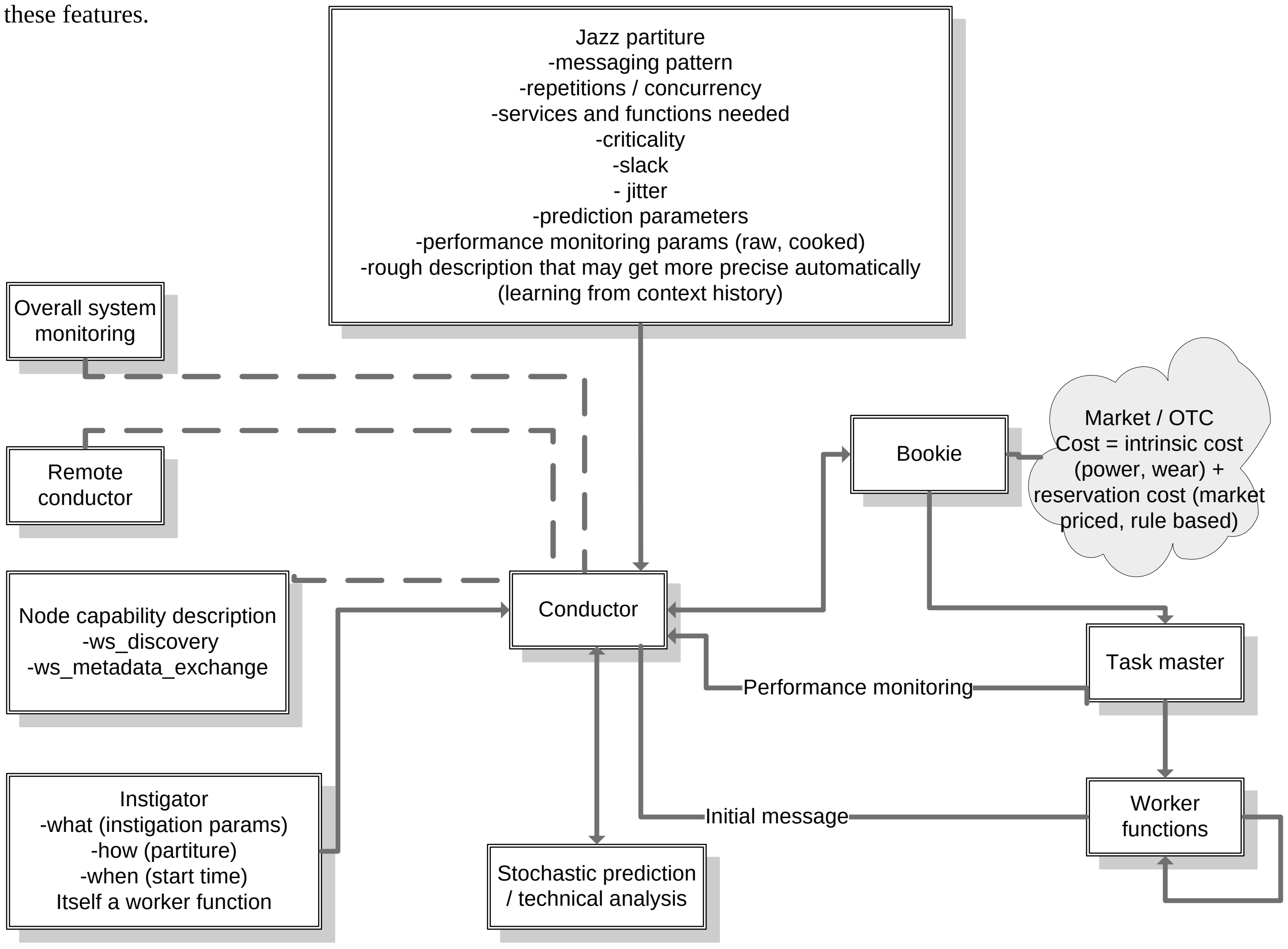
In case of stability and brake systems the same unit would be able to execute control depending of which sensors are available on a specific wheels. If appropriate sensors on wheels exist the system could execute the ABS partiture and if the engine management allows for it a stability partiture could be executed that involves the engine management. If the hardware exists for a specific feature but no partiture is available to take advantage of the existing hardware the suitable partiture could be purchased from the manufacturer and loaded to the car. With the low costs of hardware such a scenario is quite feasible where all the hardware features are built into the car, but enabled only when the appropriate partiture exists to make use of these features.

The owners could purchase new features from the manufacturer's web page and the features would easily be loaded to the car by the owner himself. These features could include engine management which would give more power, better audio system features, enabling preloaded navigation system features, etc. In principle users could subscribe to new features just like subscribing to a TV channel. Case in point, the author's pickup truck has the same engine as a larger model. Getting an extra 100 horse powers would entail replacing the engine ROM manually. Instead filling a form on a web site would be convenient.

The access and modification privileges should be strictly controlled but for some features the consumer could have access directly. For other features access would be limited to car rental companies, for some features to authorized dealers only and some to any maintenance shop. Depending of the type of user the car could also provide different information. For example engine operation parameters provided to dealers could be more accurate than ones provided to consumers or generic maintenance shops.

Some information could be available to anyone, e.g. road conditions could be communicated to other cars, providing a way to warn other drivers if conditions are dangerous.

Note that communication overhead within the car is less relevant since modern automotive buses such as FlexRay offer sufficient bandwidth (up to 10 MBit/s) even for more demanding traffic.


Jazz partiture
-messaging pattern
-repetitions / concurrency
-services and functions needed
-criticality
-slack
- jitter
-prediction parameters
-performance monitoring params (raw, cooked)
-rough description that may get more precise automatically (learning from context history)
Overall system monitoring
Remote conductor
Node capability description
-ws_discovery
-ws_metadata_exchange
Instigator
-what (instigation params)
-how (partiture)
-when (start time)
Itself a worker function
Conductor
Stochastic prediction / technical analysis
Bookie
Market / OTC
Cost = intrinsic cost (power, wear) + reservation cost (market priced, rule based)
Task master
Performance monitoring
Worker functions
Initial message

ARCHITECTURE

To create a modular system which configuration and the roles of the devices is defined at runtime we propose an architecture that relies on the usage of metadata to describe (among other things) the set of functions involved in a computing scenario, the interactions between the nodes executing the functions and the approaches used for monitoring the execution of these functions. We introduce the concept of a computing partiture – a collection of metadata about a computing scenario – as the source of information for the nodes executing the scenario. In addition to describing a computing scenario the partiture also allows describing how the context information required for a computing scenario is collected and used.

The partiture does not contain details of the implementation of the functions involved in the partiture – it only describes the functions that are involved and the metadata relevant to these functions. Neither does the partiture contain information on the specific nodes that should execute the partiture but it rather describes the functions that are executed as part of the partiture. The functions described in a partiture can run on one or more nodes depending on the details of the partiture and the availability of resources at the nodes in the given network.

The partiture describes the interactions (messaging patterns) between the functions including the timing constraints of the individual interactions – intervals of execution, mean, slack and jitter. In addition the possible repetitions and repetition intervals of the partiture are described.

The conductor

To execute the partiture every node contains a conductor that can execute a partiture. The conductor is responsible for selecting the nodes that are going to execute the functions described in the partiture and delivering the information required for the execution to the nodes. The conductor sends the relevant part of the instantiated partiture to each node participating in the execution of the partiture. The segment of the partiture sent to the participating node contains information on what functions or services should be executed, timing constraints of the functions, messaging patterns, context collection and analysis. A conductor is also responsible for making agreements with conductors on other nodes to perform their part of the partiture. The conductor executes the model described by the partiture, predicts and plans.

Interoperability

The messaging used in our system is built around XML Web Services [7] taking maximum advantage of decades of interoperation efforts in business and more recently e-business and web services based management protocols, such as WS-Management. The conductor uses WS-Discovery to discover participating nodes. If some of the devices do not support the conductor and partitures, they can still be incorporated but real-time response will be less orderly. This will be accounted for in the schedule as instrumentation will observe random delays from the non-real-time components and pad the schedule accordingly.

A CASE STUDY

The claim that even quite thin embedded nodes are able to perform the predictions on context parameters is not unsubstantial, since in [3] as well as in the case study presented in this paper it is shown how even quite simple mathematical models suffice to predict the future values of context parameters, such as execution times of scheduled functions, with quite good results.

Implementation

The concept of applying simple stochastic methods on predicting context information was experimented on a test platform equipped with a 25 MHz Arm7 microcontroller with 256KB of ROM and 32KB of RAM. In the core of the study was a stochastic planner that used the monitored execution times of scheduled functions to make adjustments to the scheduling pattern of the functions.

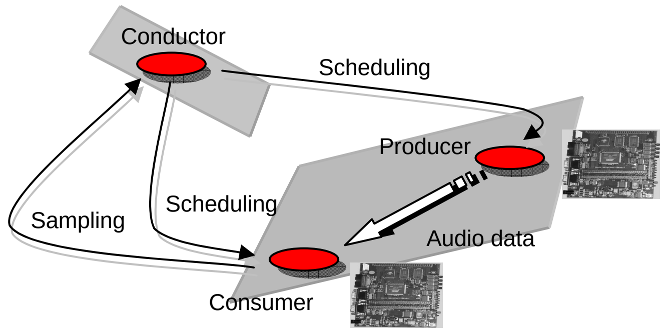


Figure 2. Scheme of the stochastic planner in action

Initially the planner uses an application supplied fixed schedule for scheduling the jobs on the worker nodes. The schedule is adjusted according to the information on the actual execution times received from worker nodes.

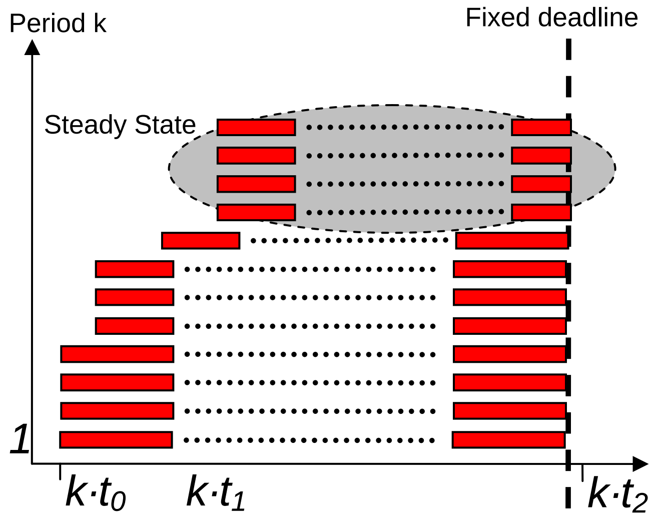


Figure 3. Adaptation of function executions from application supplied defaults to a near optimal steady state

PERFORMANCE

The working of the stochastic planning conductor was estimated through sampling. A simple test method does 20000 multiplications. Starting with no context information the conductor uses an application provided guess.

Once the conductor receives samples from the measured execution times it uses the information with smoothing between each step. The calculation times include formatting and sending the reply message. The table below contains the relevant numbers. The estimate is produced by the live conductor, while the mean and deviation have been calculated offline for reference from the raw measurements.

Step	Estimate 95% conf	Measured mean	Standard deviation	Confidence 95%	99%
1	339	337	1.7%	1.0	1.4
2	341	337	1.6%	1.0	1.4
3	346	337	1.8%	1.0	1.4

Figure 4. Time measurement and prediction of a CPU intensive task – times in milliseconds, 32 samples per iteration on embedded microcontroller board. The confidence number indicates the extra time allocated for jitter. Fixed point integer arithmetic rounds the number up slightly.

Since the low-level RTOS scheduler did not produce much jitter, the test was also executed on a PC running Windows XP with the XML communications middleware stack on top. Running without an underlying real-time scheduler introduces more uncertainty but the conductor still deals with it correctly and produces a larger confidence allocation to cope with the increased jitter. As the CPU is faster a million multiplications is done each time. From a steady state the number of calculations is dropped to half. The table below shows how the prediction adapts to the drop. The conductor adapts to the larger jitter by padding the estimates.

Step	Estimate 99% conf	Measured mean	Standard deviation	Confidence 95%	99%
1	126	123	6.4%	1.9	2.5
2	124	120	14%	4.2	5.5
3	69	55	2.1%	2.8	3.7
4	58	55	2.9%	3.9	5.2

Figure 5, Time measurement on PC in milliseconds. After the steady state at step 2, the workload is cut in half and the estimate adapts to the new load.

CONCLUSION

This paper introduced a programming framework for writing adaptive distributed applications for future automobiles. Distributed processed are described in a *partiture* that describes a feature or task as a pattern of communicating devices and functions. The partitures allow putting together car components and expose features selectively. New partitures can be purchased and downloaded from the manufacturer's web site.

Instrumentation, as directed by the partiture produces a context history. This is fed into a stochastic process that automatically adapts real-time tasks to changes in the configuration of the vehicle as well as changes in the environment. Embedded XML Web Services provides interoperation within and outside the automobile.

REFERENCES

1. Stankovic J, Ramamritham K, "The Spring Kernel: A New Paradigm for Real-Time Systems", IEEE Software, Volume 8 , Issue 3 , pp 62 - 72 May, 1991.
2. Broy, M, "Challenges in automotive software engineering", International Conference on Software Engineering Proceeding of the 28th international conference on Software engineering, pp 33 – 42. 2006.
3. FlexRay Consortium. http://www.flexray.com/
4. Grimm, K, "Software technology in an automotive company: major challenges" , International Conference on Software Engineering Proceedings of the 25th International Conference on Software Engineering, pp 498 – 503. 2003.
5. Hardung, B, Kölzow, T, Krüger, A, "Distributed systems: Reuse of software in distributed embedded automotive systems", Proceedings of the 4th ACM international conference on Embedded software EMSOFT '04 pp 203 – 210. 2004.
6. Helander J., Sigurdsson S., Self-Tuning Planned Actions: Time to Make Real-Time SOAP Real, Proceedings of the Eighth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05) - Volume 00, 2005.
7. Helander J., Deeply Embedded XML Communication: Towards an Interoperable and Seamless World, Proceedings of the 5th ACM int. conf. on Embedded software, Sept. 18-22, Jersey City, NJ, USA, 2005.
8. Mamaysky H., Lo A. W. and Wang J. Foundations of technical analysis: Computational algorithms, statistical inference, and empirical implementation. Journal of Finance, 40(4), August 2000.
9. Nolte, T, Hansson, H, Bello, L, "Implementing Next Generation Automotive Communications", Proceedings of the 1st Embedded Real-Time Systems Implementation Workshop (ERTSI'04) in conjunction with the 25th IEEE International Real-Time Systems Symposium (RTSS'04), Lisbon, Portugal, 2004.
10. Preden J, Helander, J, "Auto-adaptation Driven by Observed Context Histories", 2nd International Workshop on Exploiting Context Histories in Smart Environments – Infrastructures and Applications. Newport Beach, CA, USA, 2006.