The Parking Assignment Problem
Software Engineering Mini Project Report 1

Gurupungav Narayanan
16CO114, Computer Engineering
National Institute of Technology Karnataka
gurupungavn@gmail.com

Nihal Haneef
16CO128, Computer Engineering
National Institute of Technology Karnataka
nihalh55@gmail.com

***Abstract*—In a world where time is valued more than ever, where appointments need to happen on time and deadlines need to be met, finding a parking slot for your car only wastes time and fuel. The search for a parking place consumes a lot of time and affects the efficiency of economic activities, social interactions, and the health of citizens. An ideal parking space needs to be as close as possible to the drivers destination, while conserving fuel and time as much as possible. An absence of sufficient parking lots results in drivers using undesignated areas for parking, resulting in traffic blocks and congestion. Our proposed solution is a software that would automatically detect open slots, and would assign the most ideal free slot to a driver. The software also needs to be able to detect and adapt to various layouts, and be able to ensure that the driver only parks at the designated spot.**

I. Real Time Software

For the proposed parking assignment software to be actually implemented we would need sensors to detect the presence of cars at each slot and return values to the control software. This brings to focus, the real-time aspect of our proposed solution.

A. Software Development Models

After a survey and review conducted on existing software design methods for real time systems, [1] concludes that all existing methods have some limitation in terms of real time systems design. [1] proposes a new approach to real time systems design called DARTS (Design Approach for Real Time Systems). [2] discusses another model for real-time systems called COMET. COMET is a Concurrent Object Modeling and Architectural Design Method for the development of concurrent applications, in particular distributed and real- time applications. The COMET Object-Oriented Software Life Cycle is highly iterative. These models have similar application cases and can be used to build a real-time centric solution to our proposed problem. [9] discusses the use of an assertive specification language for real-time software development and maintenance. It is used for asserting the facts or relations inherent in the problem to be solved, as opposed to conventional programming languages, which are used to express the computer solution.

B. UML and Formal notations for Real Time Software

[3] explores a dual approach to real-time software development. Models are first written in UML, as this is expected to be relatively easy and economic. Then, models are automatically translated into a formal notation that supports the verification of properties such as safety, utility, liveness, etc. In this way, developers can exploit the advantages of formal notations while skipping the complex and expensive formal modelling phase. The proposed approach is applied to the Generalised Railroad Crossing (GRC) problem. [3] highlights the shortcomings of UML as a real-time modelling language, proposes enhancements and workarounds to overcome UML limitations, and demonstrates the viability of using UML as a front-end for a formal real-time notation. Some elements of this work can be used in our proposed software solution.

II. Component-Based Software Development

Our proposed software can be easily split into modules and sub-modules that can be considered to be independent components. These modules applications need not be limited to our solution and can be expanded to be used elsewhere. For instance, OCR to detect number plates of cars can have further applications that can be used for say, over speeding detection in highways.

A. Engineering component-based software

Good Software Engineering practices have been centered around a divide and conquer approach. The approach has been to divide software into manageable components, while maximizing cohesion and minimizing coupling between components. Dividing software into components is economical; especially when there are Commercial-Off-The-Shelf modules that already exist that perform the required function. This way, we can extend from existing libraries and save cost and time. Components can be seen as sub-softwares that are capable of independent execution. This makes debugging and error correction easier, while making the code exportation simple. [4] proposes a reference model for a systematic approach to building component based systems. The model focuses mainly on technical aspects of the component assembly process.

B. Aspect Oriented Component Requirements Engineering

Developing requirements for software components, and ensuring these requirements are met by component designs, is very challenging, as often application domain and stakeholders are not fully known during component development. [5] introduces a new methodology, aspect-oriented component engineering, that addresses some difficult issues of component requirements engineering by analyzing and characterizing components based on different aspects of the overall application a component addresses. Each of these aspects would

have their own aspect details from which the overall software requirements can be derived. This makes, requirements engineering for component based systems easier.

C. Software Testing using Aspect Oriented Programming

[10] examines the suitability of Aspect Oriented Programming for the purpose of performing various types of software testing. Aspect Oriented Programming modularizes the cross-cutting concerns into units called aspects and separates them from the modules implementing the primary business logic. This leads to a system that is easier to understand and simpler to maintain. The basis of the idea behind using Aspect Oriented Programming for software testing is that aspects in Aspect Oriented Programming can be used to capture execution points within the program's modules and this can be used to test components where bugs are expected without even modifying the source code.

D. Model for Component-based systems verification

Component oriented programming has a lot of advantages as discussed before. However, using off-the-shelf components safely and properly is a challenge, which signifies the need to verify the components, to assure the systems correctness. [6] proposes a formal model for designing and verifying component-based systems using a Maude-based engine. [6] also proposes a transformation engine generate formal specification of component-based systems.

III. Context-sensitive Software Development

Our proposed software is event-driven since every function is activated only after an event occurs. This makes the software context-sensitive in nature. For example, when a Car parks in its assigned spot results in the system closing that spot and prevents other cars from being assigned that slot. This gives us an option to incorporate context-sensitive models for software development.

A. Dynamic Condition Response Graphs

Changes in the context are naturally described as events, either provided by sensors, user inputs or messages from other programs. This suggests the application of event-driven programming to implement context-aware systems. [7] uses the recently introduced event-based declarative process model, Dynamic Condition Response Graphs (DCR Graphs) as a formal basis for modular and aspect-oriented construction of context-sensitive systems. One way to think of a DCR Graph is as an event-driven reactive program, where the code-blocks usually executed when an event pattern is detected have been replaced by specifications of response events. Basically when a car parks, leaves a parking slot, parks at wrong places, are events and how we would react to each event is shown via a DCR graph. [8] describes how the declarative Dynamic Condition Response (DCR) Graphs process model can be used for trustworthy adaptive case management by using the flexible execution, dynamic composition and adaptation supported by DCR Graphs.

References

[1] H. Gomaa. 1984. *A software design method for real-time systems*. Commun. ACM 27, 9 (September 1984), 938-949. DOI=http://dx.doi.org/10.1145/358234.358262

[2] Hassan Gomaa. 2000. *Designing real-time and distributed applications with UML*. In Proceedings of the 22nd international conference on Software engineering (ICSE '00). ACM, New York, NY, USA, 829-. DOI=http://dx.doi.org/10.1145/337180.337855

[3] Luigi Lavazza, Gabriele Quaroni, and Matteo Venturelli. 2001. *Combining UML and formal notations for modelling real-time systems*. SIGSOFT Softw. Eng. Notes 26, 5 (September 2001), 196-206. DOI=http://dx.doi.org/10.1145/503271.503236

[4] A. W. Brown and K. C. Wallnan, *Engineering of component-based systems*, Engineering of Complex Computer Systems, 1996. Proceedings., Second IEEE International Conference on, Montreal, Que., 1996, pp. 414-422. DOI=10.1109/ICECCS.1996.558485

[5] J. Grundy, *Aspect-oriented requirements engineering for component-based software systems*, Proceedings IEEE International Symposium on Requirements Engineering (Cat. No.PR00188), Limerick, 1999, pp. 84-91. DOI=10.1109/ISRE.1999.777988

[6] A. A. Debza, C. Bouanaka and N. Zeghib, *Towards rewriting-based formal model for component-based systems verification*. 2016 International Conference on Advanced Aspects of Software Engineering (ICAASE), Constantine, 2016, pp. 46-53. DOI=10.1109/ICAASE.2016.7843862

[7] Thomas Hildebrandt, Raghava Rao Mukkamala, Tijs Slaats, and Francesco Zanitti. 2013. *Modular context-sensitive and aspect-oriented processes with dynamic condition response graphs*. In Proceedings of the 12th workshop on Foundations of aspect-oriented languages (FOAL '13). ACM, New York, NY, USA, 19-24. DOI=http://dx.doi.org/10.1145/2451598.2451604

[8] R. R. Mukkamala, T. Hildebrandt and T. Slaats, *Towards Trustworthy Adaptive Case Management with Dynamic Condition Response Graphs*. 2013 17th IEEE International Enterprise Distributed Object Computing Conference, Vancouver, BC, 2013, pp. 127-136. doi: 10.1109/EDOC.2013.22

[9] J. S. Tseng, B. Szymanski, Y. Shi and N. S. Prywes. *Real-time software life cycle with the model system*. IEEE Transactions on Software Engineering, vol. SE-12, no. 2, pp. 358-373, Feb. 1986. doi: 10.1109/TSE.1986.6312949

[10] M. Jain and D. Gopalani. *Aspect Oriented Programming and Types of Software Testing*. 2016 Second International Conference on Computational Intelligence and Communication Technology (CICT), Ghaziabad, 2016, pp. 64-69. doi: 10.1109/CICT.2016.22