Title

Building a Public Transportation System Product Line

Proposed by: Timothy C. Lethbridge, University of Ottawa, Canada tcl@site.uottawa.ca

Abstract

Participants will create a product line that can be instantiated for any public transportation system (busses, underground trains etc.) to manage routes, runs, schedules, etc. They will do this using model-driven development and the Umple language to the largest extent possible,. The main purposes of the project are to explore the requirements of such systems and how they can all be embodied in one project line, as well as to exercise the Umple language and tools.

A. Introduction

I have created this project for two reasons: Firstly, the domain of public transportation is interesting, since every city in the world needs software that will embody the same basic concepts, yet each city may have its own special needs. This is therefore a very suitable topic to explore developing a product line model that can be used to automatically generate systems that meet the specific needs of each city. Secondly I would like to exercise Umple; in other words, to develop lots of applications using it so we can perfect it. Umple has been found to be quite suitable for the development of product lines [4].

Product line development involves developing not just a single application, but a model and/or code base that can be used to generate any number of applications within a domain. Each instantiation of the product line will satisfy a different set of requirements. The requirements will therefore need to be specified such that they cover all possible instantiations.

It is widely accepted that software should be developed using languages that embody abstractions at a high level. This should enable the developer to focus on solving domain and business problems rather than implementation problems. Model-driven development has this as one of its main aims. However, for various reasons many developers cling to programming as their prime means of development rather than adopting modeling. The Umple language family enables developers to use modeling concepts, such as associations and state machines, while still writing code in a form that is familiar with them. This can be described as ‘model-oriented programming’. Umple is under active development but has already been used to create a variety of systems, including the Umple compiler itself. For information about Umple, see [1], [2] and [3]. More complete details of Umple, including the Eclipse plugin will be sent to teams who contact the project sponsor as described in section D.

The two key constraints on this project are: firstly that you model a transportation product line and secondly that you use the Umple language for modeling and generating code. You may in addition use other modeling tools and development. For product-line representation we have a tool called VML4Umple [5], a tool we have also developed, but you may adapt any other product-line tool to manage Umple files if you wish.

Since Umple is under development in a research environment, you may be faced with challenges relating to occasional bugs, missing features or incomplete explanations. Helping us fix these is an integral part of this project. A rule of the project is that you should not (and should not need to) edit the Java or PHP code generated by Umple. However, if you encounter problems with Umple that you cannot work around, this rule may be temporarily broken after you have requested permission to do so.

B. Required functionality that must be available in the resulting systems

The product line should be able to instantiate systems that capture most of the following functionality. Project teams should study real public transportation systems to uncover differences among them that would lead to differences in requirements, and hence variability points in the product line.

1. **Core elements**. The system should manage stops (bus stops, stations, etc.), lines (also called routes), and particular runs (both scheduled and actual) on a line at a particular time on a particular day.

2. **Schedules and changes to schedules**. For example, there may be one set of stops, lines and runs currently, but there may be different sets active on holidays, Sundays, etc. Also, starting on a certain date the entire schedule may be changed (e.g. a new line added, frequency changed, etc.). Finally, extra runs may be added, or detours made, without notice, to deal with emergency situations.

3. **Timetables and passenger assistance**. The system will need to be able to generate timetables for the use of passengers. This would include tables of all the runs on each route on a given day, and also guidance about how to get from one stop to another, including transfers.

4. **Actual timing**. Some instantiations of the system should be able to track the *actual* location of a vehicle on a run, and be able to compute the time it is *projected* to arrive at each stop. This projected time should ideally be taken into account in item 3 above.

5. **Vehicles assigned to runs or groups of runs**. By vehicles we mean specific trains or busses. The system will need to handle replacements when breakdowns occur as well as scheduled maintenance. It is suggested that early iterations of the project omit this item.

C. Other general project parameters

1. Demonstration using a variety of instantiations

Each project team should be able to demonstrate that its product line can generate instantiations that work well for at least 3 or 4 public transportation systems that differ from each other in interesting ways. At least one of these should be for a real city; the others could be hypothetical.

3. Extending the requirements

There are certain optional aspects of public transportation systems that you may choose to model, but you may also choose to omit these. You may ignore them entirely, or only activate them for a subset of your instantiations. Examples of optional aspects include ticketing, booking on reserved seats, scheduling of staff to drive the vehicles, charters, etc.

You will be given credit for uncovering and dealing with requirements that have not been specified in this document. Demonstrating that you have explored a wide range of requirements will be key to success in this project.

4. Dealing with Umple problems

If you discover problems in Umple or the Umple tools, it would be nice if you could contribute to fixing those bugs in an open-source manner, but doing so is not a requirement of the project.

5. Architectural issues: Persistence and User Interface

The results of this project should be considered prototypes intended to explore requirements. It would be quite sufficient for *persistence* to be managed by simply loading and dumping memory

using something like JBoss, and to have a 'terminal' user interface whose commands are entered through standard input. There is therefore no explicit requirement to develop sophisticated persistent storage mechanism (e.g. databases), nor a graphical user interface. That said, there is an Umple tool for generating a test user interface automatically, and any extensions to provide persistence and UI facilities will be counted when judging the results. Any UI or persistence layers should be enabled using a layered approach so the Umple-generated business logic does not need to be modified and does not have unnecessary dependencies with other layers.

6. Process model

Development should be performed in an agile but disciplined manner. In particular, development should proceed in the smallest possible increments, and test-driven development is mandatory. After every increment there should be a demonstrable and tested executable system.

7. Final Deliverables (as a zip file)

a) A detailed report in pdf format containing at least the following

i. An abstract, an introduction and a background section.
ii. Diagrams, where possible generated from the code in item b below, with textual captions describing each diagram.
iii. A record of the development process followed. This should include details what was done and created at each iteration.
iv. Instructions regarding how to install and use the resulting system.
v. A description of the experiences of the development team. This should include lessons learned and suggested improvements to the Umple language tools, etc.
vi. Conclusions

b) Well-documented files containing model-oriented product line source code. This will consist of Umple code plus directives for generating an appropriate subset of the Umple code for each instantiation of the product line. Inside the Umple source code will be model elements like classes, associations, attributes and state machines, plus Java (or PHP) embedded methods and statements. All descriptions of the models should be embedded in the code using extensive comments.

c) The executable test cases used during development.

8. Judging criteria

a) Quality and comprehensiveness of the product-line requirements for the public transportation domain and its instantiations. This includes innovative and interesting requirements, demonstrations of important differences in requirements for different instantiations, as well as resolution of conflicts among requirements.

b) Correct, comprehensive and high-quality use of Umple, including model elements, comments and embedded Java or PHP code for methods and conditions

c) Adherence to the agile process as demonstrated by the test cases, the delivery of interim reports (see below) and the description of experiences in the final report.

d) Quality of writing of the final report

Bonuses may also be given for:

e) Other extensions to the project, such as persistence, user interface, etc.

f) Contributions to improving Umple that we adopt, whether it be actual code changes or well-thought-out suggestions for improvements.

D. Interaction between stakeholder and developing teams

1. Each team interested in this project should contact the project sponsor at tcl@site.uottawa.ca The subject line in the Email should be “SCORE 2011 request for Umple”.

2. The project sponsor will make available on the web or via email the latest build of Umple and its related tools and documentation (particularly the Eclipse plugin). This is not currently openly available on the Internet simply because it has been undergoing change and we like to keep track of who is using Umple so we can advise them of updates.

3. Each team will be advised to next explore Umple by creating simple test systems before embarking on the main project.

4. Any questions about *how to use Umple, or reports of problems or bugs*, should be addressed to the project sponsor by email. The project sponsor will most likely ask a member of his team to respond to the question, with responses to be returned within 3 business days. A FAQ will be created so that multiple teams with the same question do not overburden the team. Questions already answered in the FAQ might simply receive the response ‘See the FAQ’ if the volume of questions becomes too high.

5. Any questions about *interpretation of this document (i.e. what is required in the project),* should also be addressed to the project sponsor by email. The sponsor will endeavor to respond to each request within 3 business days. As above, an FAQ will be created.

6. Up to three times during the course of the project, each team is invited to send a draft of the report to the project sponsor. The sponsor will look at each draft and provide feedback. If a very large number of teams participate, the feedback may be minimal, for example just a couple of sentences giving suggestions, but may be more extensive if the project sponsor has the time and there are not too many teams participating. The sponsor will endeavor to respond within 5 business days, although during July, August and early September 2010, the sponsor may have one or more periods in which he is unavailable for up to 3 weeks.

References

[1] Umple home page: http://cruise.site.uottawa.ca/umple/

[2] Umple online: http://cruise.site.uottawa.ca/umpleonline/

[3] Forward, A., Lethbridge, T.C., and Brestovansky, D. (2009), “Improving Program Comprehension by Enhancing Program Constructs: An Analysis of the Umple language”, *International Conference on Program Comprehension (ICPC) 2009*, Vancouver, IEEE Computer Society 311-312.

[4] Levin, Jenya (2009), “System Generation for Time and Activity Management Product Lines”, Masters thesis, http://www.site.uottawa.ca/~tcl/gradtheses/jlevin/

[5] VML for Umple http://cruise.site.uottawa.ca/umpleonline/vml.html