3
Trabalhos Relacionados

Sistemas de tempo real, software embarcados, sistemas multi-agentes e *Design by Contract* não são assuntos novos e pode-se encontrar diversos trabalhos sobre os mesmos na literatura. No entanto, poucos trabalhos envolvem a utilização conjunta dessas tecnologias, como é o caso da proposta deste trabalho.

Dentre esses trabalhos pode-se citar [44] em que Marinescu fala sobre a possibilidade de se utilizar agentes de software em um software embarcado que possua comportamento ativo ou autônomo, tendo como foco a capacidade de mobilidade dos agentes. No sistema desenvolvido nesta dissertação, o foco está na autonomia dos agentes. Também pode-se citar [45] em que Richling fala sobre o desenvolvimento de sistemas embarcados de tempo real, utilizando uma arquitetura baseada em componentes que usa *Design by Contract* para definir as interfaces dos componentes, incluindo os requisitos não funcionais (tempo, recurso, etc.). No sistema desenvolvido nesta dissertação, DBC não é apenas utilizado em nível de componentes (na camada mais externa), e sim, em nível de classes (de forma mais pontual). É importante ressaltar que nenhum dos trabalhos pesquisados menciona algum projeto ou estudo de caso que tenha utilizado tais abordagens.

Em relação ao uso de assertivas em C++ pode-se destacar alguns trabalhos: Guerreiro [46, 47] e Rosenblum [24].

Em [46, 47], Guerreiro sugere um modo de se utilizar assertivas em C++ sem fazer uso de macros ou pré-processamento por considerá-las ferramentas de baixo nível. Essa abordagem possui algumas desvantagens em relação à abordagem utilizada nesta dissertação (que faz uso de macros):

– Uma assertiva é testada usando uma função específica, o que implica em invocar uma função e conseqüentemente criar todas as variáveis que a função necessita (como a mensagem de erro), antes de testar se as assertivas estão ou não ligadas e antes de testar se a condição da assertiva é verdadeira ou falsa. Na abordagem utilizada nesta dissertação, graças ao

uso de macros, caso a condição da assertiva seja verdadeira nenhum código extra é executado e nenhuma variável extra é criada.

– Toda informação acerca do erro ocorrido (em caso de uma assertiva falhar) deve ser passada como parâmetro para a função que testa a condição da assertiva. Na abordagem utilizada nesta dissertação, várias informações acerca do erro (como o nome e a linha do arquivo, e a *string* representando exatamente o teste que falhou) são anexadas automaticamente à mensagem de erro (através de valores e funções embutidos na macro) e não precisam ser passadas para a macro responsável pela assertiva.

– É impossível desligar totalmente as assertivas (fazer com que o código das assertivas seja ignorado durante a execução), mesmo com as assertivas desabilitadas, ainda há um overhead da chamada aos métodos. Na abordagem utilizada nesta dissertação, é possível desligar totalmente as assertivas fazendo uso de parâmetros de compilação.

– Guerreiro concentra as funções relacionadas às assertivas em uma única classe que deve ser estendida pelas classes que precisam usar assertivas. Isto significa que o código existente, que não seja OO (módulos em C, por exemplo), precisa ser alterado para poder incluir assertivas. Outro problema é que em muitos casos será necessário usar herança múltipla, cuja utilização já provou ser fonte de problemas em sistemas OO.[48] No entanto, esse problema pode ser contornado transformando os métodos dessa classe em funções globais (ou em métodos públicos e estáticos).

– Quando uma assertiva falha, uma exceção é levantada. Tal abordagem possui algumas desvantagens inerentes à própria linguagem de programação C++, que não possui coleta automática de memória alocada dinamicamente e não utilizada, tornando o uso de exceções uma possível fonte de vazamento de memória. Além disso, obriga a alteração de código existente para, em pontos estratégicos, tratar eventuais exceções que antes nunca ocorriam. A identificação de tais pontos pode se mostrar um trabalho dispendioso. No entanto, este problema pode ser contornado substituindo o código que levanta uma exceção por outro que tenta se recuperar do erro ou que aborta o programa.

Em [24], Rosenblum sugere um modo de utilizar assertivas em C++ fazendo uso de anotações (comentários) que são pré-processados antes da compilação e geram o código das assertivas. Esta abordagem tem a vantagem de deixar bem separado o código relacionado às assertivas do resto do código (uma vez que o código das assertivas fica dentro de comentários próprios). A possibilidade de usar uma abordagem como a sugerida por Rosenblum

foi descartada, pois verificou-se que o pré-processamento adicionado à etapa de compilação abriria possibilidade para problemas e contratempos que não ocorreriam usando apenas macros, tais como:

- A necessidade de lidar com erros de compilação que são oriundos do pré-processamento, e que por vezes são bem difíceis de detectar, pois mesmo que o pré-processador valide a sintaxe antes do pré-processamento, é possível que um código aceito pelo pré-processador acabe gerando um código inválido.
- Em alguns casos o pré-processamento pode fazer tantas alterações no código fonte original que fica impossível acompanhar a execução do programa em um depurador a partir do código fonte original (antes do pré-processamento).
- O uso do pré-processamento implica não só em um aumento da duração do tempo de compilação como um todo (tempo que não pode ser desprezado em sistemas com grande número de classes, como o caso desse estudo), como também em alguns casos acaba tornando necessário recompilar um número maior de arquivos quando o código é alterado e se deseja recompilar apenas os arquivos alterados (e aqueles que dependem diretamente deles).
- Como já havia sido decidido que o Qt seria usado no desenvolvimento do sistema, usar uma abordagem como essa implicaria em realizar dois pré-processamentos antes da compilação (uma vez que o código que utiliza certas funcionalidades do Qt precisa ser pré-processado), o que provavelmente aumentaria o impacto dos problemas relatados acima.
- No caso específico da abordagem de Rosenblum, como todo o texto que é pré-processado fica dentro de comentários, o programa pode ser compilado (por descuido, por exemplo) sem o pré-processamento (e conseqüentemente sem assertivas).
- Muitos programadores não gostam de escrever comentários (mesmo que esses comentários depois gerem código), ou pelo menos se sentem mais confortáveis escrevendo código do que comentários. Se a forma de usar assertivas não for simples, ou se o programador não se sentir confortável para fazer uso delas, chega-se à situação em que o programador começa a encarar o uso de assertivas como um trabalho burocrático, e não como algo que o auxilia, o que diminuiria a pré-disposição à aceitação da técnica por parte da equipe.
- O código pode ser alterado e o programador pode se esquecer de alterar os comentários correspondentes.

Além desses trabalhos existem algumas implementações para assertivas em C++ (a maior parte delas usando macros), mas a maior parte dessas implementações eram muito simples e não atendiam às necessidades do projeto (como a macro *assert*[49] que faz parte das funções básicas de C), ou eram muito complexas (como a Nana[50]) e por isso demandariam um tempo considerável para que se entendesse seu funcionamento e a customizasse para atender as necessidades do projeto. Por isso optou-se por uma implementação própria que, no entanto, não foi feita do zero, e se baseou na implementação sugerida em [51], que propõe um modo simples e funcional de fazer uso de assertivas utilizando macros.