

Construction and Analysis of Software Systems

(draft 0.9)

Luís Caíres e João C. Seco

Departamento de Informática

FCT/UNL

Esta nota visa complementar a ficha da unidade curricular da disciplina de “Construction and Analysis of Software Systems”, proposta no âmbito da reestruturação do segundo ciclo de Engenharia Informática do DI para a especialidade de “Software Construction and Analysis: Languages and Techniques”, de modo a melhor explicar os seus objectivos através da apresentação de algum contexto adicional.

Keywords: Programming-in-the-Large, Software Construction, Software Validation, Static Analysis, Lightweight Specification and Validation

O desenvolvimento de software de grande escala requer a construção de subsistemas robustos, abertos à reutilização e composição. A unidade curricular **Construction and Analysis of Software Systems** ensina os princípios, técnicas e ferramentas envolvidos na **construção** e **validação** de sistemas de software baseados em componentes, de forma aplicada às plataformas de suporte à execução standard de indústria (os frameworks Java e .NET).

A abordagem seguida pretende fazer a ponte útil entre técnicas frequentemente abordadas numa perspectiva mais teórica e a sua utilização prática, na actividade de construção de software. O foco da disciplina é na construção de software, usando mecanismos avançados de estruturação e validação de código.

Os temas tratados são

(1) Mecanismos e técnicas de suporte à abstracção modularidade e composição, a sua correcta utilização, e o seu suporte nas linguagens de programação industriais.

Estes mecanismos são fundamentais para suportar a construção de software baseada em componentes, mas a sua correcta utilização requiere o uso de metodologias específicas. Tal como o saber programar no detalhe (*programming-in-the-small*) não se resume ao conhecimento de uma linguagem de programação, saber estruturar um sistema no concreto (*programming-in-the-large*) não se limita ao conhecimento de um ou outro framework de componentes, mas requer familiaridade com técnicas de estruturação de software bem específicas, suportadas por padrões de estruturação e idiomas de programação conhecidos. Entre alguns destes padrões incluem-se os padrões de desenho arquitectural mais comuns. A disciplina aborda estas técnicas no contexto real da construção de software, usando ambientes de programação concretos, e de forma orientada para a integração de sistemas heterogéneos (por exemplo, sistemas envolvendo bases de dados e sistemas baseados em serviços, como os aplicados aos sistemas de gestão de informação).

(2) Técnicas de raciocínio sistemático sobre sistemas de software baseados em componentes, aplicados às linguagens de programação de uso comum (Java, C, C#).

A verificação de software, ou seja a certificação de que uma componente de software satisfaz certos requisitos (por exemplo, de correcção funcional, de uso

de recursos, de segurança) requer o uso de técnicas para sistematizar o raciocínio sobre programas. Em geral, tais técnicas baseiam-se em lógicas de programas, uma linha desenvolvida desde a década de 70, que sempre se considerou promissora mas que só mais recentemente começa a ter real impacto tecnológico, tendo entrado no mainstream da indústria de software.

“Things like even software verification, this has been the Holy Grail of computer science for many decades but now in some very key areas we’re building tools that can do actual proof about the software and how it works in order to guarantee the reliability” (Bill Gates, 2002)

Algumas técnicas de verificação poderosas baseadas no uso de asserções, utilizáveis a vários graus de profundidade no código desenvolvido, estão presentemente disponíveis ao mesmo nível que os sistemas de tipos, que são a forma de asserção antiga (tão poderosa como transparente) e banalizada nas linguagens de programação modernas (Java, C#, C). O interesse nas linguagens de asserções aumenta muito quando as mesmas podem ser **validadas estaticamente**, isto é, em tempo de compilação, usando compiladores certificadores (ESC/Java, JML, Spec#). Obtém-se então garantias sobre a correcção dos programas que vão muito além das garantidas pelos sistemas de tipos usuais, mas com um peso notacional semelhante ao requerido pelos sistemas de tipos. Em certos casos, é possível obter garantias fortíssimas, tais como ausência de erros de runtime (incluindo os erros “null object reference”, ausência de corridas, e ausência de “memory leaks”). Recentemente, uma ferramenta de análise estática industrial (ASTRÉE) foi usada para garantir a completa ausência de erros de execução no módulo de pilotagem fly-by-wire do AIRBUS A340.

“In Nov. 2003, ASTRÉE was able to prove completely automatically the absence of any RTE in the primary flight control software of the Airbus A340 fly-by-wire system, a program of 132,000 lines of C analyzed in 1^h20 on a 2.8 GHz 32-bit PC using 300 Mb of memory.” (Astreé web site)

Ora o uso correcto destas ferramentas requer a compreensão de certos formalismos lógicos e saber utilizá-los no contexto das linguagens de programação usuais (por exemplo, na determinação de invariantes de classe, nas especificações da semântica de interfaces de componentes, etc). A disciplina **Construction and Analysis of Software Systems** desenvolve competências nesta área, de forma integrada com actividades práticas de construção de software. O “nível zero” da especificação ao nível da programação concreta, é o nível da especificação de tipos, que qualquer aluno de primeiro ciclo conhece. Esta disciplina fornece competências acrescidas, ensinando o aluno a utilizar linguagens de asserções mais expressivas, baseadas em linguagens lógicas simples, para a definição de propriedades de interesse. Para que tal seja possível, serão usadas técnicas de especificação leve (lightweight specification), as quais permitem ao programador adoptar um estilo de especificação sustentável.

(3) As técnicas de análise estática ao nível de código fonte, abordando os princípios de análise e transformação de programas e as abordagens à segurança baseadas nas linguagens de programação.

Para favorecer a correcta utilização de ferramentas de análise estática, e desenvolver competências de concepção de novas ferramentas dessa natureza, é

necessário conhecer as técnicas básicas de análise estática de programas, nomeadamente análise de fluxo de dados e informação (data flow analysis), análise de fluxo de controle, interpretação abstracta e sistemas especiais e tipos e efeitos. Estas técnicas estão consolidadas e encontram aplicações na detecção de incorrecções dos programas que são cada vez mais importantes num contexto de software extensível. Entre as aplicações mais importantes podem-se referir o código certificado (proof-carrying code) e a verificação de bytecode, do género efectuado pela JVM, até à detecção de fugas de memória (memory leaks) e informação. Estas últimas, dado a sua implicação na segurança e robustez das aplicações são particularmente relevantes. O estudo dos princípios subjacentes a estas técnicas dotarão o aluno de competências acrescidas de utilização de ferramentas de análise estática, tais como compiladores certificadores já referidos (ESC/Java, JML, Spec#), de compreensão das suas limitações, assim como de concepção de algoritmos específicos de análise estática.

No fim do curso, o aluno deverá ser capaz de conceber, construir e validar um sistema de software de alguma complexidade, utilizando de forma integrada os princípios e técnicas abordadas no curso. O aluno deverá produzir uma solução de modularização do sistema, exprimi-la correctamente na linguagem de programação concreta seleccionada, justificar o seu desenho com base em padrões e métodos sistemáticos, evitando soluções *ad-hoc*. Deverá também ser capaz de especificar e validar várias propriedades de correcção nas linguagens de asserções usadas, e de as validar usando as ferramentas necessárias. Tipicamente tal sistema de software será um pequeno sistema de gestão de informação, com suporte de informação persistente e requisitos de correcção local e global.

Disciplinas com programas semelhantes, ou abordando os temas referidos, eventualmente com ênfases diferentes podem ser encontradas em algumas universidades de referência (BRICS (DE), Imperial College (UK), Pisa (IT), Carnegie-Melon (USA)).

A bibliografia base da disciplina inclui as seguintes referências:

(1) B. Liskov and J. Guttag. *Program Development in Java: Abstraction, Specication, and Object-Oriented Design*. Addison-Wesley, 2000 (Capítulos 1,3,5,9).

(2) Ghezzi, Jayazeri and Mandrioli. *Fundamentals of Software Engineering*. Prentice-Hall, 2002 (Capítulos 4).

(3) *Principles of Program Analysis*, by Flemming Nielson, Hanne Riis Nielson, and Chris Hankin. Springer-Verlag, 1999. 2000 (Capítulos 1,2).

As ferramentas exercitadas incluem:

ESC/Java2 (<http://secure.ucd.ie/products/opensource/ESCJava2/>)

Java Modeling Language (<http://www.cs.iastate.edu/~leavens/JML/>)

Spec # (<http://research.microsoft.com/specsharp/>)

Java PathFinder (<http://javapathfinder.sourceforge.net/>)