GUIDE

Graphical User Interface Development Environment

Andrea Longoni - 13896A

INTERNSHIP END DATE
08/01/25

1 VoiSmart

The work for this thesis was conducted at VoiSmart, an Italian company specializing in research and development of communication and mobility solutions. VoiSmart focuses on transforming advanced technologies into practical tools that enhance productivity and efficiency in business environments. With a strong emphasis on research, the company develops innovative products in the field of fixed and mobile telecommunications. Its approach prioritizes technological excellence, modular design, and high-quality support services.

Throughout the project, VoiSmart provided valuable guidance in aligning development choices with real-world industry needs. Their expertise in Elixir influenced the selection of target technologies, while their extensive experience in UI design helped refine both the DSL's expressiveness and the structure of the generated interfaces.

2 Context and Initial Situation

Graphical User Interface (GUI) development plays a crucial role in modern software engineering, yet existing approaches often require developers to manually reimplement similar UI structures across different frameworks and programming languages. This leads to duplicated code, increased maintenance effort, and reduced flexibility when adapting applications to new requirements. Many UI development tools are tightly coupled to specific frameworks, making modifications and extensions complex and time-consuming. The need for a more modular and automated approach to UI generation, capable of minimizing manual effort while ensuring flexibility and adaptability, served as the driving force behind this work.

3 Objectives

The primary objective of this work was to design and implement a system that simplifies UI development by introducing a high-level abstraction through a Domain-Specific Language (DSL) [4] and ensuring modularity through Software Product Line (SPL) [1] techniques. The project aimed to create a DSL that enables developers to define UIs independently of specific frameworks, a code generation system capable of producing equivalent interfaces in multiple target languages, and an SPL-based architecture that allows for selective feature inclusion and extensibility. The ultimate goal was to evaluate the effectiveness of this approach in reducing development effort, improving maintainability, and ensuring reusability across different platforms.

4 Work Performed

To achieve these objectives, the GUIDE (Graphical User Interface Development Environment) system was developed, integrating a DSL for UI definition, an SPL-based modular structure, and a code generation framework that supports multiple output languages, specifically Python's Tkinter, HTML with Bootstrap, and Elixir Phoenix LiveView. The work involved defining the GUIDE DSL's syntax and semantics using Neverlang [2, 3, 5, 6], implementing a library responsible for managing UI components and layouts, and developing language-specific adapters to handle code generation. A modular architecture was established using Gradle, ensuring that developers could configure GUIDE to include only the

necessary components, thereby reducing unnecessary complexity. Additionally, the system was evaluated by analyzing the reduction in manual coding effort and its impact on development efficiency.

5 Technologies Used

Several technologies were employed in the development of GUIDE. The core system was implemented in Java 17, leveraging Neverlang for DSL definition. Gradle was used as the build system to manage modularization and dependencies. The generated UI code targeted Python's Tkinter, HTML with Bootstrap, and Elixir's Phoenix LiveView, ensuring a diverse set of supported platforms. These technologies were chosen to balance flexibility, portability, and ease of integration with existing development workflows.

6 Results and Lessons Learned

The evaluation of GUIDE demonstrated its ability to streamline UI development by reducing the amount of manually written code and improving modularity. The generated UI structures successfully replicated the intended designs across multiple languages, confirming the system's effectiveness in automating the development process. The SPL-based modularization approach allowed for selective feature inclusion, although its impact on storage efficiency was limited due to the significant size of external dependencies. Despite these advantages, some challenges remain. GUIDE does not generate full callback implementations, requiring developers to manually handle event-driven logic. Additionally, dependency management, particularly related to Neverlang, affects the overall storage footprint, highlighting the need for further optimization.

Overall, this work demonstrated that integrating DSLs and SPLs for UI development provides a flexible and efficient solution that enhances code reusability and maintainability. Future improvements could focus on optimizing dependency management, extending GUIDE to additional UI frameworks, and developing a visual UI editor to further simplify the design process.

References

[1] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer, April 2013.

[2] Walter Cazzola. Domain-Specific Languages in Few Steps: The Neverlang Approach. In Thomas Gschwind, Flavio De Paoli, Volker Gruhn, and Matthias Book, editors, *Proceedings of the 11th International Conference on Software Composition (SC'12)*, Lecture Notes in Computer Science 7306, pages 162–177, Prague, Czech Republic, 31st of May-1st of June 2012. Springer.

[3] Walter Cazzola and Edoardo Vacchi. Neverlang 2: Componentised Language Development for the JVM. In Walter Binder, Eric Bodden, and Welf Löwe, editors, *Proceedings of the 12th International Conference on Software Composition (SC'13)*, Lecture Notes in Computer Science 8088, pages 17–32, Budapest, Hungary, 19th of June 2013. Springer.

[4] Martin Fowler and Rebecca Parsons. *Domain Specific Languages*. Addison Wesley, September 2010.

[5] Edoardo Vacchi and Walter Cazzola. Neverlang: A Framework for Feature-Oriented Language Development. *Computer Languages, Systems & Structures*, 43(3):1–40, October 2015.

[6] Edoardo Vacchi, Diego Mathias Olivares, Albert Shaqiri, and Walter Cazzola. Neverlang 2: A Framework for Modular Language Implementation. In *Proceedings of the 13th International Conference on Modularity (Modularity'14)*, pages 23–26, Lugano, Switzerland, 22nd-25th of April 2014. ACM.

GUIDE

Graphical User Interface Development Environment

Andrea Longoni - 13896A

DATA DI TERMINE DELLO STAGE
08/01/25

1 VoiSmart

Il lavoro per questa tesi è stato svolto presso VoiSmart, un'azienda italiana specializzata nella ricerca e sviluppo di soluzioni per la comunicazione e la mobilità. VoiSmart si concentra sulla trasformazione delle tecnologie avanzate in strumenti pratici per migliorare la produttività e l'efficienza negli ambienti aziendali. Con una forte attenzione alla ricerca, l'azienda sviluppa prodotti innovativi nel campo delle telecomunicazioni fisse e mobili. Il suo approccio privilegia l'eccellenza tecnologica, il design modulare e servizi di supporto di alta qualità.

Durante lo sviluppo del progetto, VoiSmart ha fornito un contributo fondamentale nell'orientare le scelte tecnologiche verso esigenze concrete del settore. La loro esperienza con Elixir ha influenzato la selezione delle tecnologie di destinazione, mentre la loro competenza nel design di interfacce utente ha contribuito a migliorare l'espressività del DSL e la struttura delle interfacce generate.

2 Contesto iniziale

Lo sviluppo di interfacce grafiche (GUI) rappresenta un aspetto cruciale dell'ingegneria del software moderna, ma gli approcci attuali spesso richiedono agli sviluppatori di reimplementare manualmente strutture UI simili in diversi framework e linguaggi di programmazione. Questo porta a duplicazione del codice, maggiori sforzi di manutenzione e minore flessibilità nell'adattare le applicazioni a nuovi requisiti. Inoltre, molti strumenti per lo sviluppo di UI sono strettamente legati a framework specifici, rendendo complesse e dispendiose le modifiche e le estensioni. La necessità di un approccio più modulare e automatizzato, capace di ridurre il lavoro manuale garantendo flessibilità e adattabilità, è stata la motivazione principale di questo lavoro.

3 Obiettivi del lavoro

L'obiettivo principale di questa tesi è progettare e implementare un sistema che semplifichi lo sviluppo delle interfacce utente, introducendo un'astrazione ad alto livello tramite un Domain-Specific Language (DSL) [4] e garantendo la modularità mediante tecniche di Software Product Line (SPL) [1]. Il progetto si propone di creare un DSL che permetta agli sviluppatori di definire le UI indipendentemente dai framework specifici, un sistema di generazione automatica del codice capace di produrre interfacce equivalenti in diversi linguaggi target, e un'architettura SPL che consenta l'inclusione selettiva delle funzionalità e l'estensibilità. L'obiettivo finale è valutare l'efficacia di questo approccio nel ridurre l'impegno di sviluppo, migliorare la manutenibilità e garantire la riusabilità del codice su diverse piattaforme.

4 Descrizione lavoro svolto

Per raggiungere questi obiettivi, è stato sviluppato il sistema GUIDE (Graphical User Interface Development Environment), che integra un DSL per la definizione delle UI, una struttura modulare basata su SPL e un framework di generazione del codice in grado di produrre interfacce in diversi linguaggi, tra cui Python con Tkinter, HTML con Bootstrap ed Elixir con Phoenix LiveView. Il lavoro ha previsto la definizione della sintassi e della semantica del DSL di GUIDE utilizzando Neverlang [2, 3, 5, 6], l'implementazione di una libreria per la gestione dei componenti UI e dei layout, e lo sviluppo di adattatori

specifici per ciascun linguaggio al fine di supportare la generazione del codice. L'architettura modulare è stata realizzata con Gradle, permettendo agli sviluppatori di configurare GUIDE includendo solo i componenti necessari e riducendo così la complessità complessiva. Inoltre, il sistema è stato valutato analizzando la riduzione dello sforzo di codifica manuale e il suo impatto sull'efficienza dello sviluppo.

5 Tecnologie coinvolte

Nel sviluppo di GUIDE sono state impiegate diverse tecnologie. Il sistema principale è stato implementato in Java 17, utilizzando Neverlang per la definizione del DSL. Gradle è stato adottato come sistema di build per gestire la modularizzazione e le dipendenze. Il codice UI generato è compatibile con Python con Tkinter, HTML con Bootstrap ed Elixir con Phoenix LiveView, garantendo il supporto a una varietà di piattaforme. Queste tecnologie sono state scelte per bilanciare flessibilità, portabilità e facilità di integrazione nei flussi di sviluppo esistenti.

6 Competenze e risultati raggiunti

La valutazione di GUIDE ha dimostrato la sua capacità di semplificare lo sviluppo delle interfacce utente, riducendo la quantità di codice scritto manualmente e migliorando la modularità. Le strutture UI generate hanno riprodotto con successo i design previsti in più linguaggi, confermando l'efficacia del sistema nell'automatizzare il processo di sviluppo. L'approccio modulare basato su SPL ha permesso l'inclusione selettiva delle funzionalità, sebbene il suo impatto sull'efficienza dello storage sia stato limitato a causa delle dimensioni significative delle dipendenze esterne. Nonostante questi vantaggi, rimangono alcune sfide. GUIDE non genera implementazioni complete dei callback, richiedendo agli sviluppatori di gestire manualmente la logica degli eventi. Inoltre, la gestione delle dipendenze, in particolare quelle legate a Neverlang, incide sullo spazio di archiviazione complessivo, evidenziando la necessità di ulteriori ottimizzazioni.

In generale, questo lavoro ha dimostrato che l'integrazione di DSL e SPL nello sviluppo di UI rappresenta una soluzione flessibile ed efficiente, migliorando la riusabilità e la manutenibilità del codice. Miglioramenti futuri potrebbero concentrarsi sull'ottimizzazione della gestione delle dipendenze, sull'estensione di GUIDE ad altri framework UI e sullo sviluppo di un editor visuale per semplificare ulteriormente il processo di progettazione.

Riferimenti bibliografici

[1] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines*. Springer, April 2013.

[2] Walter Cazzola. Domain-Specific Languages in Few Steps: The Neverlang Approach. In Thomas Gschwind, Flavio De Paoli, Volker Gruhn, and Matthias Book, editors, *Proceedings of the 11th International Conference on Software Composition (SC'12)*, Lecture Notes in Computer Science 7306, pages 162–177, Prague, Czech Republic, 31st of May-1st of June 2012. Springer.

[3] Walter Cazzola and Edoardo Vacchi. Neverlang 2: Componentised Language Development for the JVM. In Walter Binder, Eric Bodden, and Welf Löwe, editors, *Proceedings of the 12th International Conference on Software Composition (SC'13)*, Lecture Notes in Computer Science 8088, pages 17–32, Budapest, Hungary, 19th of June 2013. Springer.

[4] Martin Fowler and Rebecca Parsons. *Domain Specific Languages*. Addison Wesley, September 2010.

[5] Edoardo Vacchi and Walter Cazzola. Neverlang: A Framework for Feature-Oriented Language Development. *Computer Languages, Systems & Structures*, 43(3):1–40, October 2015.

[6] Edoardo Vacchi, Diego Mathias Olivares, Albert Shaqiri, and Walter Cazzola. Neverlang 2: A Framework for Modular Language Implementation. In *Proceedings of the 13th International Conference on Modularity (Modularity'14)*, pages 23–26, Lugano, Switzerland, 22nd-25th of April 2014. ACM.