POLITECHNIKA POZNAŃSKA

WYDZIAŁ INFORMATYKI I TELEKOMUNIKACJI
Instytut Informatyki

Sprawozdanie z realizacji projektu

AI - WYKONANIE ZADANIA (HELLO WORLD DLA JAVACARD)

PRZEDMIOT: SYSTEMY AUTOMATYCZNEJ IDENTYFIKACJI

Szymon Kupis, 151587

Prowadzący zajęcia
Marek Gosławski

POZNAŃ 2026

Spis treści

Rozdział 1

Wstęp i założenia projektowe

W dzisiejszych czasach wykorzystanie dużych modeli językowych (LLM) w procesie tworzenia oprogramowania staje się standardem. Niniejszy projekt weryfikuje ich przydatność w specyficznym, niskopoziomowym środowisku kart mikroprocesorowych JavaCard.

Celem pracy jest realizacja i wdrożenie podstawowego apletu typu „Hello World" z wykorzystaniem modeli AI (Gemini 3.1 Pro oraz Claude Opus 4.8) w roli asystentów programowania. Zakres prac obejmuje przygotowanie środowiska symulacyjnego, implementację apletu w oparciu o komunikację APDU (Application Protocol Data Unit) oraz wykonanie testów porównawczych.

Struktura pracy jest następująca. W rozdziale 2 opisano kroki przygotowania środowiska oraz implementację techniczną. Rozdział 3 zawiera szczegółową analizę rozwiązań wygenerowanych przez modele LLM oraz logi z symulatorów. Rozdział 4 stanowi podsumowanie i ocenę przydatności modeli.

Rozdział 2

Metodyka badawcza i implementacja bazowa

W tym rozdziale przedstawiono teoretyczne i techniczne aspekty realizacji zadania oraz dokładną metodykę pracy z modelami językowymi.

2.1 Założenia eksperymentu i prompt bazowy

Aby rzetelnie porównać możliwości modeli Gemini 3.1 Pro oraz Claude Opus 4.8, przyjęto metodykę badawczą opartą na strategii „zero-shot". Oznacza to, że ocenie podlegało wyłącznie pierwsze wygenerowane rozwiązanie, bez stosowania dodatkowych podpowiedzi (kolejnych promptów) korygujących ewentualne błędy.

Obu modelom dostarczono identyczne polecenie w języku angielskim, co minimalizuje ryzyko halucynacji technologicznych wynikających z tłumaczenia pojęć. W prompcie nałożono rygorystyczne wymagania dotyczące zachowania standardów platformy JavaCard oraz wykluczono zbędne komentarze w kodzie.

Poniżej zamieszczono dokładną treść promptu bazowego wykorzystanego w badaniu:

Listing 2.1: Prompt bazowy wykorzystany do wygenerowania apletu

```
You are an expert JavaCard programmer and smart card systems specialist. Your task
    is to deliver a complete "Hello World for JavaCard" project from scratch,
    following professional standards.

Implement the following points in order:

## 1. Environment Setup
Provide concise, bulleted instructions for:
* Installing Java Development Kit (JDK 8 or compatible)
* Setting up JavaCard Development Kit (JCDK)
* Configuring a simulator environment (JCIDE or equivalent)
* Verifying the toolchain is operational

## 2. Applet Implementation
Write complete, production-ready source code for a JavaCard Applet that:
* Extends `javacard.framework.Applet`
* Implements proper lifecycle methods (`install()`, `process()`)
* Follows JavaCard memory optimization practices
* Uses appropriate exception handling (ISOException)

## 3. APDU Communication Handler
```

```
Implement the `process()` method to:
* Receive and interpret incoming APDU commands
* Handle applet selection (SELECT command with proper AID)
* Route commands to appropriate handlers
* Manage response buffering

## 4. Hex Logic for "Hello World"
Implement logic that:
* Responds to a dedicated APDU command with "Hello World" in hexadecimal (48656C6C6F20576F726C64)
* Properly manages the response buffer and length
* Returns status word 9000 (success)

## 5. Testing & Validation
Provide exact APDU commands for testing:
* SELECT command (with AID and expected response)
* Command to trigger "Hello World" response
* Expected hex output including status word 9000
* Verification steps for simulator

## 6. Project Deliverables
Provide:

Source Code Structure:
* Complete file organization ready for GitHub/GitLab publication
* Build configuration (Ant/Maven if applicable)
* All source files with minimal comments (only for non-obvious byte operations)

Technical Documentation (Markdown):
* Project overview and objectives
* Architecture and design decisions
* Installation and build instructions
* API reference for the applet
* Testing procedures with examples
* Format suitable for publication on mcp.poznan.pl

Report Skeleton (Markdown):
* Executive summary template
* Project scope and requirements
* Implementation details section
* Results and testing outcomes
* Conclusions and future improvements
* Format ready for mcp.poznan.pl publication

---
Quality & Language Requirements:
* Language: Generate all explanations, step-by-step instructions, documentation, and reports in Polish. However, keep all source code, variable names, APDU commands, and technical terminology in English.
* Write concise, direct code and documentation.
* Use comments only for complex byte-level operations.
* Verify JavaCard standards compliance (memory efficiency, exception handling, APDU protocol).
* Ensure all code is production-ready and well-structured.
```

2.2 Konfiguracja środowiska

Zgodnie z informacjami wygenerowanymi przez modele, aby umożliwić tworzenie oprogramowania dla kart JavaCard, wykorzystano środowisko Java Development Kit w wersji 8 (zapewniające niezbędną wsteczną kompatybilność z narzędziami JCDK) oraz zintegrowane środowisko JCIDE.

2.3 Komunikacja APDU i aplet JavaCard

Zaimplementowany aplet dziedziczy po klasie `javacard.framework.Applet`. Komunikacja między terminalem a kartą odbywa się za pomocą jednostek APDU. Odbiór i interpretacja poleceń ma miejsce w metodzie `process()`. Aplet analizuje bajty `CLA` (klasa) oraz `INS` (instrukcja) i na ich podstawie zwraca ciąg w formacie szesnastkowym (HEX).

Rozdział 3

Testy i analiza modeli LLM

Do realizacji zadania wykorzystano identyczny prompt w dwóch środowiskach: Gemini 3.1 Pro oraz Claude Opus 4.8. Ten rozdział dokumentuje jakość dostarczonych wyników i logi z testów symulacyjnych.

3.1 Wyniki: Gemini 3.1 Pro

Model Gemini 3.1 Pro prawidłowo wygenerował instrukcje konfiguracyjne, zwracając uwagę na krytyczny wymóg wstecznej kompatybilności i konieczność użycia JDK 8 zdefiniowanego przez zmienną środowiskową `JAVA_HOME`. Podał również jasne kroki budowania aplikacji oparte na pliku `build.xml` (Apache Ant).

Struktura wygenerowanego przez model kodu apletu jest zgodna z pryncypiami platformy:

- Implementacja posiada prawidłowo nadpisaną metodę `install`, gdzie dokonano rejestracji ze zmiennej wielkości bajtami środowiskowymi `bArray`.
- Tablica znaków ASCII reprezentująca ciąg "Hello World"została statycznie alokowana w pamięci EEPROM jako zmienna `final`, co chroni przed błędami przekroczenia rozmiaru sterty podczas cyklu pracy apletu (brak alokacji w pętli obsługi komend).
- Kopiowanie bufora wyjściowego realizowane jest przez metodę `Util.arrayCopyNonAtomic`, która unika narzutu pamięciowego i cykli CPU związanego z mechanizmem transakcji w platformie JavaCard.

Poniżej zamieszczono kluczowy fragment logiki APDU wygenerowany przez model:

Listing 3.1: Obsługa APDU przez model Gemini 3.1 Pro

```
1  public void process(APDU apdu) {
2      if (selectingApplet()) {
3          return;
4      }
5
6      byte[] buffer = apdu.getBuffer();
7
8      if (buffer[ISO7816.OFFSET_CLA] != 0x00) {
9          ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
10     }
11
12     if (buffer[ISO7816.OFFSET_INS] == INS_HELLO) {
13         short le = apdu.setOutgoing();
14
```

```
        if (le < (short) HELLO_WORLD.length) {
            ISOException.throwIt(ISO7816.SW_WRONG_LENGTH);
        }

        apdu.setOutgoingLength((short) HELLO_WORLD.length);
        Util.arrayCopyNonAtomic(HELLO_WORLD, (short)0, buffer, (short)0, (short)
            HELLO_WORLD.length);
        apdu.sendBytes((short)0, (short) HELLO_WORLD.length);
    } else {
        ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}
```

Model poprawnie założył użycie testowego identyfikatora AID i wylistował adekwatne wektory komunikacyjne.

- **Wybór apletu (SELECT):** `00 A4 04 00 0A A0 00 00 00 62 03 01 0C 01 01` → Oczekiwana odpowiedź: `90 00`
- **Wywołanie logiki aplikacji (HELLO):** `00 00 00 00 0B` → Oczekiwana odpowiedź: `48 65 6C 6C 6F 20 57 6F 72 6C 64 90 00`

Model poprawnie odrzucił również zbędne instrukcje CLA, chroniąc przed błędem `SW_CLA_NOT_SUPPORTED`. Ustalona długość wiadomości wychodzącej (zmienna `Le`) była precyzyjnie kontrolowana, rzucając wyjątek w przypadku braku pełnego zadeklarowania wymaganego zasobu z bufora do zwrotu wiadomości. Wygenerowana dokumentacja techniczna oraz szkielet raportu spełniły format gotowy do publikacji.

3.2 Wyniki: Claude Opus 4.8

Model prawidłowo zinterpretował założenia projektowe i samodzielnie wygenerował gotową strukturę plików (m.in. `build.xml` dla Apache Ant, pliki źródłowe oraz dokumentację). Claude wykazał się również znajomością specyfiki środowiska, słusznie wskazując na konieczność użycia kompilatora JDK w wersji 8, co jest wymogiem konwertera `ant-javacard`.

Wygenerowana struktura projektu:

```
SAI/
|-- README.md
|-- build.xml
|-- .gitignore
|-- src/com/helloworld/HelloWorldApplet.java
|-- test/apdu-commands.md
|-- docs/
    |-- DOCUMENTATION.md
    |-- REPORT.md
```

W zakresie optymalizacji pamięci model zastosował metodę `Util.arrayCopyNonAtomic`, eliminując narzut transakcyjny. Prawidłowo zidentyfikowano również mechanizm środowiska JCRE, które automatycznie dołącza kod statusu `90 00` przy użyciu metody `setOutgoingAndSend`.

Listing 3.2: Implementacja apletu wygenerowana przez Claude Opus 4.8

```
public void process(APDU apdu) {
    if (selectingApplet()) {
        return;
    }

    byte[] buffer = apdu.getBuffer();

    if (buffer[ISO7816.OFFSET_CLA] != CLA_HELLO) {
        ISOException.throwIt(ISO7816.SW_CLA_NOT_SUPPORTED);
    }

    switch (buffer[ISO7816.OFFSET_INS]) {
        case INS_HELLO:
            sendHelloWorld(apdu);
            break;
        default:
            ISOException.throwIt(ISO7816.SW_INS_NOT_SUPPORTED);
    }
}

private void sendHelloWorld(APDU apdu) {
    byte[] buffer = apdu.getBuffer();
    short length = (short) HELLO_WORLD.length;

    Util.arrayCopyNonAtomic(HELLO_WORLD, (short) 0, buffer, (short) 0, length);
    apdu.setOutgoingAndSend((short) 0, length);
}
```

Zaproponowane wektory testowe APDU do weryfikacji w symulatorze:

- **Wybór apletu (SELECT):** `00 A4 04 00 07 01 02 03 04 05 00 01 00` → `90 00`
- **Wywołanie ciągu szesnastkowego (HELLO):** `80 40 00 00 00` → `48 65 6C 6C 6F 20 57 6F 72 6C 64 90 00`

3.3 Testy działania w symulatorze

Zgodnie z rygorem badawczym projektu, opartym na analizie wyników wygenerowanych z pojedynczego promptu bazowego, do weryfikacji działania obu apletów wykorzystano dostarczone przez modele wektory APDU i przeprowadzono klasyczną symulację.

3.3.1 Symulacja dla apletu Gemini 3.1 Pro

Poniżej przedstawiono logi z komunikacji dla parametrów wygenerowanych przez model Gemini:

Listing 3.3: Log APDU z symulatora dla apletu Gemini

```
CLA: 80, INS: ba, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 0b, 0a, a0, 00, 00, 00, 62, 03, 01, 0c, 01,
    01, Le: 00, SW1: 90, SW2: 00
CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 0a, a0, 00, 00, 00, 62, 03, 01, 0c, 01, 01,
    Le: 00, SW1: 90, SW2: 00
CLA: 00, INS: 00, P1: 00, P2: 00, Lc: 00, Le: 0b, 48, 65, 6c, 6c, 6f, 20, 57, 6f,
    72, 6c, 64, SW1: 90, SW2: 00
```

3.3.2 Symulacja dla apletu Claude Opus 4.8

Model Claude zaproponował w swojej odpowiedzi możliwość wygenerowania automatycznych testów (jCardSim/JUnit), jednak w ramach zachowania spójności metodyki badawczej (brak dodatkowych promptów sterujących), przetestowano podstawowe komendy przedstawione w pierwotnym rozwiązaniu.

Listing 3.4: Log APDU z symulatora dla apletu Claude

```
1 CLA: 80, INS: ba, P1: 00, P2: 00, Lc: 00, Le: 00, SW1: 90, SW2: 00
2 CLA: 80, INS: b8, P1: 00, P2: 00, Lc: 09, 07, 01, 02, 03, 04, 05, 00, 01, 00, Le:
      07, 01, 02, 03, 04, 05, 00, 01, SW1: 90, SW2: 00
3 CLA: 00, INS: a4, P1: 04, P2: 00, Lc: 07, 01, 02, 03, 04, 05, 00, 01, Le: 00, SW1:
      90, SW2: 00
4 CLA: 80, INS: 40, P1: 00, P2: 00, Lc: 00, Le: 0b, 48, 65, 6c, 6c, 6f, 20, 57, 6f,
      72, 6c, 64, SW1: 90, SW2: 00
```

Rozdział 4

Podsumowanie i wnioski

Praca nad projektem pozwoliła na praktyczne zestawienie środowiska JavaCard ze zdolnościami wiodących modeli językowych.

Analiza wykazała, że model Claude Opus 4.8 poradził sobie nieco lepiej z wygenerowaniem gotowej struktury projektu oraz logiki apletu, prawidłowo implementując optymalizację pamięci oraz mechanizmy środowiska JCRE. Z kolei Gemini 3.1 Pro popełnił błąd pomijając niezbędną komendę INSTALL w skrypcie APDU. Niemniej jednak, żaden z modeli w trybie "zero-shot"nie potrafił wygenerować w 100 procentach poprawnego syntaktycznie skryptu APDU dla archaicznego narzędzia apdutool z pakietu JCDK 2.2.1 (oba modele miały problem z rygorystycznym określaniem długości zmiennej Lc). Obaj asystenci znacząco przyspieszyli proces badawczy, jednak wymagały stałego i uważnego nadzoru programisty, szczególnie w fazie testowania i omijania ograniczeń starego symulatora.

Dodatek A

Materiały dodatkowe

A.1 Repozytorium kodu

Cały kod źródłowy wygenerowany w ramach projektu oraz pliki konfiguracyjne zostały opublikowane w otwartym repozytorium: `https://github.com/markowaro/SAI`