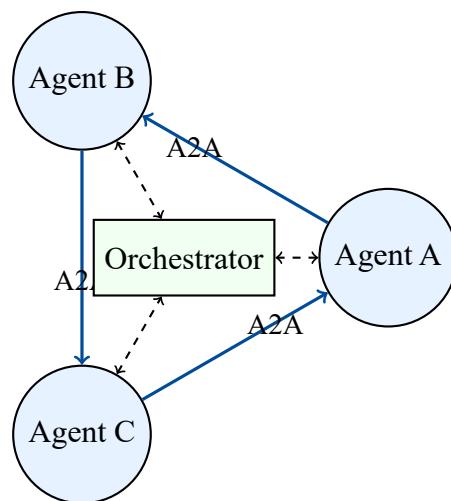


פרק 6

פרוטוקול A2A -- כשלוכנים מדברים ביניהם

איור 1.6 מציג את הרעיון המרכזי של הפרק: סוכנים אוטונומיים המתקשרים ביניהם באמצעות פרוטוקול מובנה, תחת תיאום של מתזמר (Orchestrator) מרכזי.



איור 1.6: מערכת Multi-Agent עם מתזמר מרכזי

יעדי הלמידה

בסיום פרק זה תוכלו:

- להבין מהו פרוטוקול A2A ואיך סוכנים מתקשרים זה עם זה
- לתכנן ארכיטקטורה של מערכת Multi-Agent
- לנוהל תיאום ותיזמון בין סוכנים אוטונומיים
- להתמודד עם קונפליקטים והתנגשויות בין סוכנים
- להשתמש בכלים תזמור (Orchestration) כמו LangGraph
- לנטר ולמדוד ביצועים של מערכות Multi-Agent

1.6 מבוא: הכוון של רבים

חברת מסחר אלקטרוני גדולה עומדת בפני עצמה: כל הזמנה שמנעה דורשת טיפול של מספר מחלקות -- שירותים צריך לאשר את הפרטיהם, מחלוקת המלאי צריכה לבדוק זמינות, מחלוקת התמחור צריכה לאשר מחירים והנחהות, ולוגיסטיקה צריכה לתכנן משLOW. בעבר, כל זה נעשה באמצעות מערכות נפרדות, אימילים אינטנסיביים ושיחות טלפון. כל הזמנהלקח שעotta, לפחות מפעם לפעם.

פתרונות אפשרי היה לבנות סוכן ענק אחד שמתפל בהכל. אבל הסוכן הזה היה צריך לדעת הכל -- את כל כללי המלאי, את כל מדיניות התמחור, את כל מגבלות הלוגיסטיקה. הוא היה נהייה מורכב מדי, איטי מדי, וקשה מדי לתחזוקה.

הפתרון שהחברה בחרה היה שונה: במקום סוכן אחד גדול, היא בנתה חמישה סוכנים קטנים, כל אחד מומחה בתחומו. סוכן שירותים, סוכן מלאי, סוכן תמחור, סוכן לוגיסטיקה וסוכן תשומות. כל אחד מהם ידע לעשות דבר אחד, אבל לעשות אותומצוין. האתגר האמתי התחייב כשהחברה שאלת: איך הסוכנים האלה ידברו אחד עם השני? מי יחליט מי עושה מה? מה קורה אם שני סוכנים לא מסכימים? איך מודאים שכולם עובדים לקרהת אותה מטרה?

זהו בדיקת התפקיד של פרוטוקול (A2A) .Agent-to-Agent (A2A)

2.6 מהו A2A?

1.2.6 הגדרה

הגדרה 1.6 (פרוטוקול A2A). Agent-to-Agent Protocol (A2A) הוא פרוטוקול תקשורת המאפשר לסוכנים אוטונומיים לתקשר זה עם זה, לחלק מידע, בהתאם פועלות ולבוד ייחד להשגת מטרה מסוימת [1], [4].

בעוד ש-MCP (פרוטוקול Model Context) עוסק בתקשורת בין סוכן בודד לבין המודל שלו, A2A עוסק בתקשורת בין סוכנים [9], [01].

2.2.6 מדוע נדרש A2A?

בעולם העסקי, בעיות רבות מורכבות מדי עברו סוכן בודד:

- **מומחיות מבוזרת:** ידע שונה נמצא במקומות שונים (PRE, MRC, מלאי, לוגיסטיקה)
- **זמן ותפסה:** סוכן אחד לא יכול לטפל בהכל בו-זמןית
- **חוון (Resilience):** אם סוכן אחד נכשל, אחרים יכולים להמשיך
- **התמחות:** כל סוכן יכול להתמחות بما שהוא עושה הכי טוב
- **סקלביות:** קל יותר להוסיף סוכנים חדשים מאשר להגדיל סוכן קיים

דוגמה: מערכת הזמנות

במערכת הזמנות אונליין:

- **סוכן קבלה (Intake Agent):** מקבל את הזמנה, מודוד שנתונות תקינות
- **סוכן מלאי (Inventory Agent):** בודק זמינות מוצרים

- **סוכן מחיר (Pricing Agent)**: מחשב מחיר סופי, כולל הנוחות
 - **סוכן לוגיסטיקה (Logistics Agent)**: מתזמן משולח
 - **סוכן תשלום (Payment Agent)**: מעבד תשלום
- כל סוכן עושה את התפקיד שלו, אבל הם חייבים לתקשר כדי להשלים את הזמנה מקופה.

3.2.6 מרכיבי A2A

פרוטוקול A2A טיפוסי מורכב מהמרכיבים הבאים:

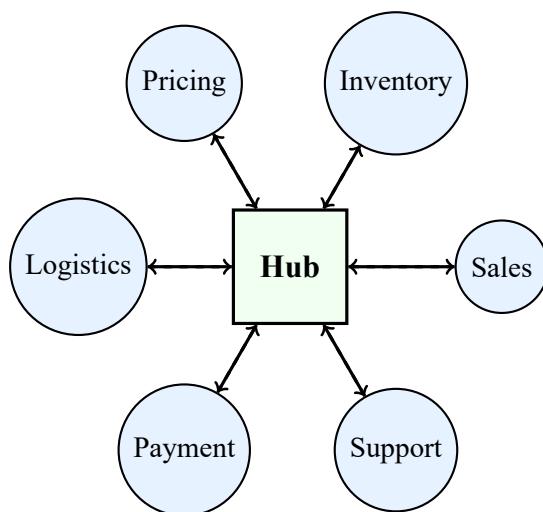
1. **שכבה תקשורת (Communication Layer)**: איז הסוכנים שולחים ומקבלים הודעות
2. **פורמט הודעות (Message Format)**: מבנה סטנדרטי של הודעות (לרוב JSON)
3. **ניטוב (Routing)**: כיצד הודעה מגיעה לסוכן הנכון
4. **תיזמור (Orchestration)**: מנגנון לתיאום פעולות בין סוכנים
5. **ניהול מצב (State Management)**: מעקב אחריו מצב השיחה/המשימה
6. **טיפול בשגיאות (Error Handling)**: מה קורה כשסוכן נכשל

3.6 ארכיטקטורות של מערכות tnegA-itluM

ישנן שלוש ארכיטקטורות עיקריות למערכות Multi-Agent:[41],[3]:

1.3.6 ekopS-dna-buH (מרכז וזרועות)

בארכיטקטורה זו, יש סוכן מרכזי אחד (Hub) שמתאים את כל האחרים (Spokes), כפי שown באיור 2.6.



אייר 2.6: ארכיטקטורת Hub-and-Spoke -- סוכן מרכזי מתאים את כולם

יתרונות:

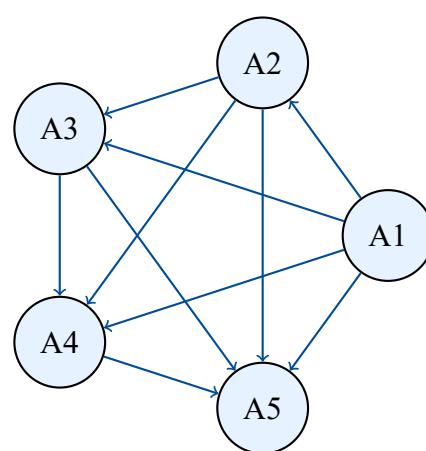
- פשוט לתוכן ולניהול
- בקרה מרכזית ברורה
- קל לדבג ולנטר

 חסרונות:

- ה-Hub הוא נקודת כשל ייחוד (Single Point of Failure)
- יכול להפוך לצוואר בקבוק (Bottleneck)
- קשה לסקלabilיות גבוהה

 2.3.6 (רשת) hseM רשת

בארכיטקטורת רשת (Mesh), כל סוכן יכול לדבר עם כל סוכן אחר ישירות, כפי שמצוג באיוור 3.6.



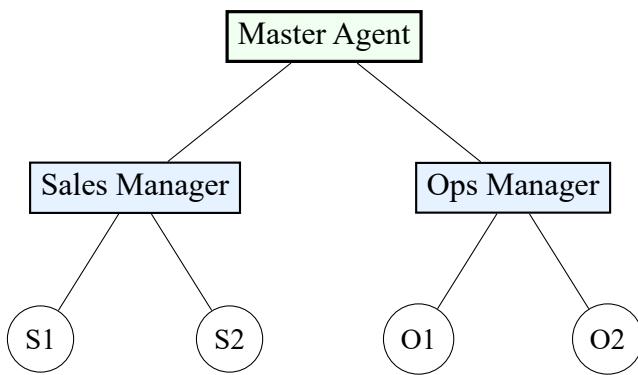
איור 3.6: ארכיטקטורת Mesh -- כל סוכן מתחבר ישירות עם כולם

 יתרונות:

- אין נקודת כשל ייחוד
- גמישות גבוהה
- תקשורת ישירה ומהירה

 חסרונות:

- מורכב מאוד לניהול
- קשה לוודא עקביות
- קשה לעקוב אחרי זרים מידע



איור 4.6: ארכיטקטורה היררכית -- סוכן ראשי מנהלי מבנים

3.3.6 lacihcraleiH (היררכי)

בארכיטקטורת היררכית, סוכנים מאורגנים בשכבות, כאשר סוכנים ברמה גבוהה יותר מנהלים סוכנים ברמה נמוכה יותר. איור 4.6 מדגים מבנה זה.

יתרונות:

- מבנה ברור וסדר
- קל לניהל באופן מודולרי
- מתאים לארגוני קיימים

חסרונות:

- יכול להיות איטי (הודעות עוברות דרך שכבות)
- נוקשה למדדי
- דורש ניהול זהיר של ההיררכיה

4.6 תיאום משימות בין סוכנים

1.4.6 חלוקת עבודה (ksaT noitubirthsiD)

אחד השאלות החשובות ביותר במערכת Multi-Agent היא: איך מחלקים עבודה בין הסוכנים? ישנן מספר אסטרטגיות:

חלוקת סטטית (tnemngissA citatS)

כל סוכן מקבל מראש סט קבוע של משימות.

דוגמה: חלוקה סטטית

- סוכן A: כל ההזדמנויות מאירופה
- סוכן B: כל ההזדמנויות מאסיה
- סוכן C: כל ההזדמנויות אמריקה

יתרונות: פשוט, ברור, צפוי

חסרונות: לאamazon (אם אירופה עמוסה ואסיה פנויה, סוכן A יהיה עמוס ואילו סוכן B יושב בטל)

חלוקת דינמית (tnemngissA cimanyD)

משימות מחולקות בזמן אמת על פי זמינות וכושר.

נוסחה: חלוקת משימות

$$\text{Task_Distribution} = \frac{\text{assigned_tasks}}{\text{max_tasks}}$$

כאשר:

- **sksat_dengissa**: מספר המשימות שהוקצו לסוכן
 - **sksat_xam**: מספר המשימות העיקרי שסוכן יכול לטפל בו במקביל
- סוכן זמין כאשר $\text{Task_Distribution} < 1.0$

יתרונות: איזון עומסים, ניצול אופטימלי,
חסרונות: מורכב יותר לישום, דורש מעקב בזמן אמת

חלוקת מבוססת יכולות (desaB-ytilibapaC)

משימות מחולקות על פי היכולות והמומחיות של כל סוכן.

דוגמה: חלוקה מבוססת יכולות

- משימה: "אשר הזמנה עם הנחת PIX"
- **סוכן מלאי:** בודק זמינות
- **סוכן ת מהיר PIX:** מחשב הנחה (בגלל שיש מומחיות ב-PIX)
- **סוכן לוגיסטי מהירה:** מותאם משלווה מהיר

2.4.6 תיאום זמינים (noitanidrooC)

כאשר מספר סוכנים עובדים על אותה משימה, חשוב להתאם ביניהם [6], [21].

סידרתי (laitneuqeS)

בתיאום סידרתי, סוכן A מסיים ועביר לsocen B, שמסיים ועביר לsocen C, כפי שown באיור 5.6.



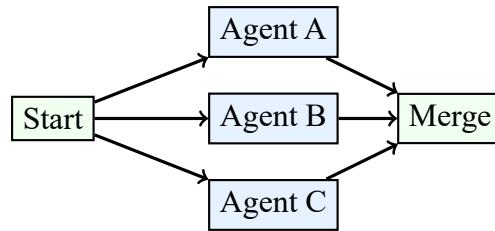
איור 5.6: תיאום סידרתי -- כל סוכן ממתיין לקודמו

יתרונות: פשוט, ברור, קל לדbg

חסרונות: איטי (כל סוכן חייב לחכות לקודם)

6.6. תיאום מקבילי (lellarP)

בתיאום מקבילי, כמה סוכנים עובדים בו-זמנית על חלקים שונים של המשימה, כפי שמודגם באירור 6.6.

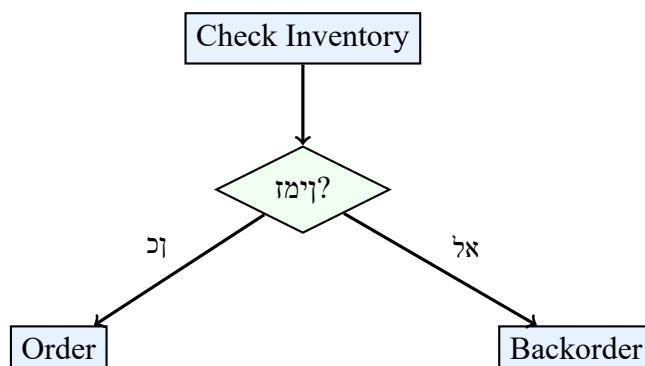


איור 6.6: תיאום מקבילי -- סוכנים עובדים בו-זמנית

יתרונות: מהיר, יעיל
 חסרונות: דורש סyncron, מורכב יותר

6.7. תיאום מותנה (lanoitidnoC)

בתיאום מותנה, הסוכן הבא תלוי בתוצאת הסוכן הנוכחי. איירור 7.6 מציג דוגמה לזרימה כזו.



איור 7.6: תיאום מותנה -- הנתיב נקבע לפי תוצאת הבדיקה

3.4.6. עלויות תיאום (O daehrev noitanidrooC)

תיאום בין סוכנים דורש זמן ומשאבים. חשוב למדוד את העלות הזו.

נוסחה: עומס תיאום

$$\text{Coordination_Overhead} = \frac{\text{coordination_time}}{\text{total_execution_time}}$$

כאשר:

הזמן שMOVED שMOVED לשירות ותיאום בין סוכנים :emit_noitanidrooc

זמן לביצוע המשימה :emit_noitucexe_latot

ערך נמוך (< 2.0) מצביע על תיאום יעיל.

ערך גבוה (> 5.0) מצביע על עומס תיאום גבוה מדי.

חישוב עומס תיאום

משמעות: עיבוד הזמן

- זמן עבודה אפקטיבי: 30 שניות
- זמן תקשורת בין סוכנים: 01 שניות
- סך הזמן: 04 שניות

$$\text{Coordination_Overhead} = \frac{10}{40} = 0.25 = 25\%$$

זהו עומס סביר. אם התקשרות היתה לוקחת 52 שניות, היינו מקבלים:

$$\text{Coordination_Overhead} = \frac{25}{55} = 0.45 = 45\%$$

זהו עומס גבוה מדי, כדאי לשקל מיזוג סוכנים או שיפור התקשרות.

5.6 טיפול בקונפליקטים

כאשר מספר סוכנים עובדים יחד, בלתי נמנע שייהיו מצבים שבהם הם לא מסכימים או שפעולותיהם מתנגשות [11], [31].

1.5.6 סוגי קונפליקטים

קונפליקט משאבים (resource contention)

שני סוכנים רוצחים להשתמש באותו משאב.

דוגמה: קונפליקט מלאי

- **סוכן הזמן A:** רוצה להקצות את הפריט האחרון במלאי ללקוח PIV
 - **סוכן הזמן B:** רוצה להקצות את אותו פריט להזמנה דחופה
- הפתרון: **צריך מגננו נעילה (Locking) או עדיפויות (Priorities).**

קונפליקט החלטה (decision contention)

שני סוכנים מגיעים להחלטות שונות על אותה בעיה.

דוגמה: קונפליקט תמחור

- **סוכן תמחור:** מחשב הנחה של 01%
- **סוכן שיווק:** מציע הנחה של 02% כחלק ממוצע

הפתרון: צרך מנגן בוררות (Arbitration) או החלטה (Arbitration).

קונפליקט סדר פעולות (tcilfnoC gniredrO)

שני שותכנים רוצים לערוך פעולה ש策ריכות להיות בסדר מסוים.

דוגמה: קונפליקט עדכון

□ סוכן A: רוצה לעדכן מחיר מוצר

□ סוכן B: רוצה לעדכן מלאי של אותו מוצר

אם שני העדכנים קוראים בו-זמנית, עלול להיווצר מצב לא עקי.

הפתרון: סנכרון (Synchronization) או עסקאות (Transactions).

2.5.6 מגנוני פתרון קונפליקטים

(עדיפות) seitiroirP

לכל שותן יש רמת עדיפות, ושותן בעדיפות גבוהה יותר "מנצח".

Listing 6.1: Python: Priority-Based Resolution

```

1 class Agent:
2     def __init__(self, name, priority):
3         self.name = name
4         self.priority = priority
5
6     def resolve_conflict(agent1, agent2, resource):
7         """Resolve resource conflict by priority."""
8         if agent1.priority > agent2.priority:
9             print(f"{agent1.name} gets {resource}")
10            return agent1
11        else:
12            print(f"{agent2.name} gets {resource}")
13            return agent2
14
15 # Example
16 vip_agent = Agent("VIP Agent", priority=10)
17 standard_agent = Agent("Standard Agent", priority=5)
18
19 resolve_conflict(vip_agent, standard_agent, "Last item")
20 # Output: VIP Agent gets Last item

```

(SFCF) devreS-tsriF-emoC-tsriF

מי שהגיע ראשון מקבל את המשאב.

(הצבעה) gnitov

מספר סוכנים "מצביעים" על הפתרון הנכון.

(הסלמה) noitalacsE

אם הסוכנים לא מסכימים, החלטה עוברת לסוכן ברמה גבוהה יותר או למנהל אנושי.

Listing 6.2: Python: Escalation Mechanism

```

1  class ConflictManager:
2      def __init__(self):
3          self.escalation_threshold = 3
4          self.conflicts = []
5
6      def handle_conflict(self, agent1, agent2, issue):
7          """Handle conflict between agents."""
8          # Try automated resolution
9          if self.can_resolve_automatically(issue):
10              return self.auto_resolve(agent1, agent2, issue)
11
12          # Escalate to human
13          self.conflicts.append({
14              'agent1': agent1,
15              'agent2': agent2,
16              'issue': issue,
17              'escalated': True
18          })
19
20          print(f"ESCALATION: {issue} needs human decision")
21          return None
22
23      def can_resolve_automatically(self, issue):
24          """Check if issue can be resolved automatically."""
25          return issue.get('auto_resolvable', False)
26
27      def auto_resolve(self, agent1, agent2, issue):
28          """Automatically resolve simple conflicts."""
29          if 'priority' in issue:
30              return agent1 if agent1.priority > agent2.priority else
agent2
31          return None

```

תיזמור סוכנים: noitartsehcro 6.6**1.6.6 מהו ?noitartsehcro**

הגדרה 2.6 (Orchestration) הוא התהליך של תיאום, ניהול וניהול של מערכת כדי להבטיח שכל הסוכנים עובדים ביעילות לקראת המטרה המשותפת [51].

תיזמור כולל:

- ניתוב הودעות בין סוכנים
- ניהול מצב השיחה
- קביעת סדר ביצוע
- טיפול בשגיאות
- מעקב אחרי התקדמות

2.6.6 כלי לתזמור: hparGgnal

nohtyP הוא ספריית המאפשרת לבנות זרימות עבודה (Workflows) מורכבות של סוכנים [7, 8].

רכיבי hparGgnal

- sedoN (צמתים): מייצגים סוכנים או פעולות
- segdE (קשות): מייצגים מעברים בין סוכנים
- etatS (מצב): המידע המשותף בין הסוכנים
- lanoitidnoC (קשותות מותניות): מעברים שתלוים בתנאי

דוגמה: wolfkroW gnissecorP redrO

Listing 6.3: Python: LangGraph Multi-Agent System

```

1 from langgraph.graph import StateGraph, END
2 from typing import TypedDict, Annotated
3 import operator
4
5 # Define shared state
6 class OrderState(TypedDict):
7     order_id: str
8     customer: str
9     items: list
10    inventory_status: str
11    price: float
12    delivery_date: str
13    payment_status: str
14    errors: Annotated[list, operator.add]
15
16 # Define agent functions
17 def intake_agent(state: OrderState) -> OrderState:
18     """Receive and validate order."""
19     print(f"Intake: Processing order {state['order_id']} ")
20     # Validation logic here
21     return state
22
23 def inventory_agent(state: OrderState) -> OrderState:
24     """Check inventory availability."""

```

```

25     print(f"Inventory: Checking stock for {state['items']}")  

26     # Check stock  

27     state['inventory_status'] = 'available'  

28     return state  

29  

30 def pricing_agent(state: OrderState) -> OrderState:  

31     """Calculate final price."""  

32     print(f"Pricing: Calculating price")  

33     state['price'] = 100.0 # Simplified  

34     return state  

35  

36 def logistics_agent(state: OrderState) -> OrderState:  

37     """Schedule delivery."""  

38     print(f"Logistics: Scheduling delivery")  

39     state['delivery_date'] = '2025-12-20'  

40     return state  

41  

42 def payment_agent(state: OrderState) -> OrderState:  

43     """Process payment."""  

44     print(f"Payment: Processing ${state['price']}")  

45     state['payment_status'] = 'completed'  

46     return state  

47  

48 # Define routing logic  

49 def should_continue(state: OrderState) -> str:  

50     """Decide next step based on state."""  

51     if state.get('inventory_status') != 'available':  

52         return 'backorder'  

53     return 'continue'  

54  

55 # Build the graph  

56 workflow = StateGraph(OrderState)  

57  

58 # Add nodes  

59 workflow.add_node("intake", intake_agent)  

60 workflow.add_node("inventory", inventory_agent)  

61 workflow.add_node("pricing", pricing_agent)  

62 workflow.add_node("logistics", logistics_agent)  

63 workflow.add_node("payment", payment_agent)  

64  

65 # Add edges  

66 workflow.add_edge("intake", "inventory")  

67 workflow.add_conditional_edges(  

68     "inventory",  

69     should_continue,  

70     {  

71         "continue": "pricing",  

72         "backorder": END  

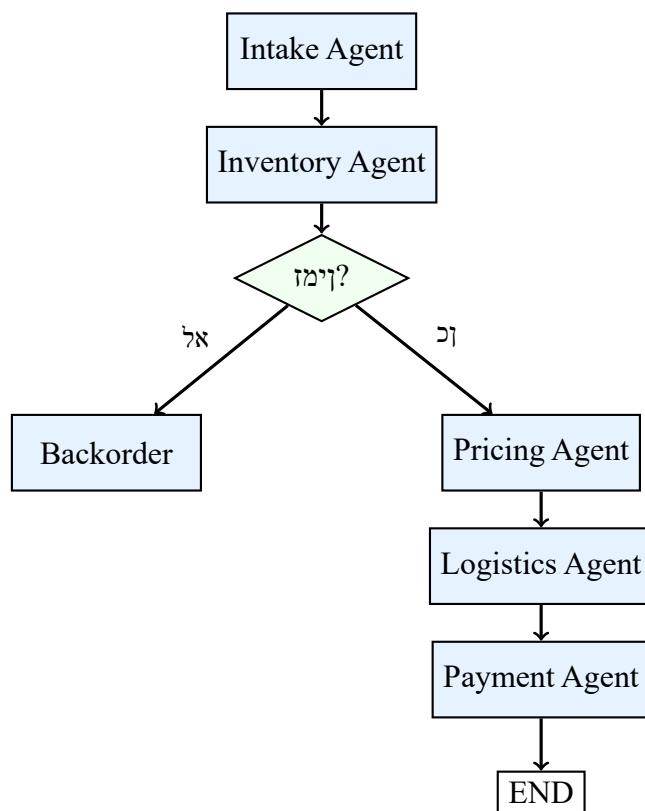
73     }  

74 )
75 workflow.add_edge("pricing", "logistics")

```

```
76 workflow.add_edge("logistics", "payment")
77 workflow.add_edge("payment", END)
78
79 # Set entry point
80 workflow.set_entry_point("intake")
81
82 # Compile
83 app = workflow.compile()
84
85 # Run the workflow
86 initial_state = {
87     "order_id": "ORD-12345",
88     "customer": "John Doe",
89     "items": ["laptop", "mouse"],
90     "errors": []
91 }
92
93 result = app.invoke(initial_state)
94 print(f"\nFinal state: {result}")
```

איור 8.6 מציג את הראשים הזרימה של המערכת.



איור 8.6: זרימת עבודה ב-LangGraph לעיבוד הזמןנות

3.6.6 ממצב (טסטים מנהליים)

אחד האתגרים הגדולים ב-Multi-Agent הוא ניהול מצב משותף.

(מצב משותף) etatS derahS

כל הסוכנים ניגשים לאותו מצב.

יתרונות: פשוט, כולם רואים את אותו מידע
 חסרונות: קונפליקטים אפשריים, דרש נעלמות

(העברת הודעות) egasseM gnissaP

כל סוכן שומר את המצב שלו, ושולח הודעות אחרים.

יתרונות: אין קונפליקטים, סוכנים עצמאיים
 חסרונות: מרכיב יותר, קשה יותר לעקוב

gnicruoS tnevE

כל שינוי רשום כאירוע, והמצב הנוכחי הוא סכום כל האירועים.

יתרונות: היסטוריה מלאה, ניתן לשחרר מצבים קודמים
 חסרונות: דרש תשתיית נוספת

7.6 ניטור מערכות M-itluA negtA**1.7.6 מטריקות חשובות**

כדי לנהל מערכת Multi-Agent ביעילות, חשוב למדוד [2], [5]:

- .1 **(תפוקה)**: כמה משימות המערכת מעבדת ליחידת זמן tuphguorhT
- .2 **(זמן תגובה)**: כמה זמן לוקח לסיים משימה מקצתה לסתה ycnetaL
- .3 **(ניצול סוכנים)**: כמה אחוזים מהזמן כל סוכן עובד noitazilitU negtA
- .4 **(שיעור שגיאות)**: כמה משימות נכשלות etaR rorrE
- .5 **(עומס תיאום)**: כמה זמן הולך על תקשורת daehrevO noitanidrooC
- .6 **(קונפליקטי משאבים)**: כמה קונפליקטים מתרחשים stclfnoC ecruoseR

2.7.6 כל ניטור**gniggoL**

תיעוד כל פעולה של כל סוכן.

Listing 6.4: Python: Agent Logging

```

1 import logging
2 from datetime import datetime
3
4 class MonitoredAgent:
5     def __init__(self, name):
6         self.name = name
7         self.logger = logging.getLogger(name)
8         self.logger.setLevel(logging.INFO)

```

```

9      # File handler
10     fh = logging.FileHandler(f'{name}_log.txt')
11     formatter = logging.Formatter(
12         '%(asctime)s - %(name)s - %(levelname)s - %(message)s'
13     )
14     fh.setFormatter(formatter)
15     self.logger.addHandler(fh)
16
17
18     def execute_task(self, task):
19         """Execute task with logging."""
20         self.logger.info(f"Starting task: {task}")
21         start_time = datetime.now()
22
23         try:
24             # Do the work
25             result = self.perform_work(task)
26
27             duration = (datetime.now() - start_time).total_seconds
28             self.logger.info(
29                 f"Completed task: {task} in {duration}s"
30             )
31             return result
32
33         except Exception as e:
34             self.logger.error(f"Failed task: {task}, Error: {e}")
35             raise
36
37     def perform_work(self, task):
38         """Actual work simulation."""
39         import time
40         time.sleep(0.5) # Simulate work
41         return f"Result of {task}"
42
43 # Usage
44 agent = MonitoredAgent("InventoryAgent")
45 agent.execute_task("check_stock_item_12345")

```

gnicarT

מעקב אחרי זרימת משימה דרך כל הסוכנים.

Listing 6.5: Python: Simple Tracing

```

1 from typing import Dict, List
2 from dataclasses import dataclass, field
3 from datetime import datetime
4
5 @dataclass
6 class TraceEvent:
7     """Single event in execution trace."""

```

```

8     timestamp: datetime
9     agent: str
10    event_type: str # 'start', 'end', 'message'
11    details: Dict
12
13 @dataclass
14 class ExecutionTrace:
15     """Complete trace of a task execution."""
16     task_id: str
17     events: List[TraceEvent] = field(default_factory=list)
18
19     def add_event(self, agent, event_type, details):
20         """Add event to trace."""
21         event = TraceEvent(
22             timestamp=datetime.now(),
23             agent=agent,
24             event_type=event_type,
25             details=details
26         )
27         self.events.append(event)
28
29     def get_duration(self):
30         """Calculate total execution time."""
31         if len(self.events) < 2:
32             return 0
33         start = self.events[0].timestamp
34         end = self.events[-1].timestamp
35         return (end - start).total_seconds()
36
37     def print_trace(self):
38         """Print execution trace."""
39         print(f"\nTrace for Task {self.task_id}")
40         print("-" * 60)
41         for event in self.events:
42             print(f"{event.timestamp.strftime('%H:%M:%S.%f')[:-3]}"
43                   f"{event.agent:15} | {event.event_type:10} | "
44                   f"{event.details}")
45         print(f"\nTotal Duration: {self.get_duration():.3f}s")
46
47 # Usage
48 trace = ExecutionTrace(task_id="ORD-12345")
49
50 trace.add_event("IntakeAgent", "start", {"action": "validate_order"})
51 trace.add_event("IntakeAgent", "end", {"status": "valid"})
52 trace.add_event("InventoryAgent", "start", {"action": "check_stock"})
53 trace.add_event("InventoryAgent", "end", {"status": "available"})
54 trace.add_event("PricingAgent", "start", {"action": "calculate"})
55 trace.add_event("PricingAgent", "end", {"price": 100.0})

```

```
56 trace.print_trace()  
57
```

draobhsaD scirteM

ויזואלייזציה של מצב המערכת בזמן אמיתי.

דשborcz מטריקות

מטריקות חשובות להציג:

- sutatS tnegA: איזה סוכנים פעילים/לא פעילים
- eueuQ ksaT: כמה שימושים ממתינות לכל סוכן
- etaR sseccuS: אחוז ההצלחה של כל סוכן
- ycnetaL egarevA: זמן ממוצע לכל סוכן
- hparG noitacinummoC: מי מדבר עם מי
- goL rorrE: שגיאות אחרונות

8.6 דוגמאות מעשיות

1.8.6 דוגמה 1: צוות סוכנים לטיפול בהזמנה

בנייה של מערכת שלמה לטיפול בהזמנות מקצועית.

Listing 6.6: Python: Complete Order Processing System

```
1 from typing import Dict, Any  
2 from dataclasses import dataclass  
3 from enum import Enum  
4  
5 class OrderStatus(Enum):  
6     NEW = "new"  
7     VALIDATED = "validated"  
8     IN_STOCK = "in_stock"  
9     OUT_OF_STOCK = "out_of_stock"  
10    PRICED = "priced"  
11    SCHEDULED = "scheduled"  
12    PAID = "paid"  
13    COMPLETED = "completed"  
14    FAILED = "failed"  
15  
16 @dataclass  
17 class Order:  
18     order_id: str  
19     customer: str  
20     items: list
```

```

21     status: OrderStatus = OrderStatus.NEW
22     price: float = 0.0
23     delivery_date: str = ""
24     payment_id: str = ""
25     error: str = ""

26
27 class OrderProcessingSystem:
28     def __init__(self):
29         self.agents = {
30             'intake': IntakeAgent(),
31             'inventory': InventoryAgent(),
32             'pricing': PricingAgent(),
33             'logistics': LogisticsAgent(),
34             'payment': PaymentAgent()
35         }
36
37     def process_order(self, order: Order) -> Order:
38         """Process order through all agents."""
39         print(f"\n{'='*60}")
40         print(f"Processing Order: {order.order_id}")
41         print(f"{'='*60}")
42
43         # Step 1: Intake
44         order = self.agents['intake'].process(order)
45         if order.status == OrderStatus.FAILED:
46             return order
47
48         # Step 2: Inventory
49         order = self.agents['inventory'].process(order)
50         if order.status == OrderStatus.OUT_OF_STOCK:
51             print("Order on backorder")
52             return order
53
54         # Step 3: Pricing
55         order = self.agents['pricing'].process(order)
56
57         # Step 4: Logistics
58         order = self.agents['logistics'].process(order)
59
60         # Step 5: Payment
61         order = self.agents['payment'].process(order)
62
63         if order.status == OrderStatus.PAID:
64             order.status = OrderStatus.COMPLETED
65             print(f"\n*** Order {order.order_id} COMPLETED ***")
66
67     return order
68
69 class IntakeAgent:
70     def process(self, order: Order) -> Order:
71         """Validate order data."""

```

```

72     print(f"\n[INTAKE] Validating order {order.order_id}")
73
74     if not order.customer:
75         order.status = OrderStatus.FAILED
76         order.error = "Missing customer info"
77         return order
78
79     if not order.items:
80         order.status = OrderStatus.FAILED
81         order.error = "No items in order"
82         return order
83
84     order.status = OrderStatus.VALIDATED
85     print(f"[INTAKE] Order validated: {len(order.items)} items")
86
87     return order
88
88 class InventoryAgent:
89     def __init__(self):
90         self.stock = {
91             'laptop': 10,
92             'mouse': 50,
93             'keyboard': 30
94         }
95
96     def process(self, order: Order) -> Order:
97         """Check inventory."""
98         print(f"\n[INVENTORY] Checking stock")
99
100        for item in order.items:
101            if item not in self.stock:
102                print(f"[INVENTORY] Item not found: {item}")
103                order.status = OrderStatus.OUT_OF_STOCK
104                return order
105
106            if self.stock[item] < 1:
107                print(f"[INVENTORY] Out of stock: {item}")
108                order.status = OrderStatus.OUT_OF_STOCK
109                return order
110
111        order.status = OrderStatus.IN_STOCK
112        print(f"[INVENTORY] All items available")
113        return order
114
115 class PricingAgent:
116     def __init__(self):
117         self.prices = {
118             'laptop': 1000.0,
119             'mouse': 25.0,
120             'keyboard': 75.0
121         }

```

```

122     def process(self, order: Order) -> Order:
123         """Calculate price."""
124         print(f"\n[PRICING] Calculating price")
125
126         total = sum(self.prices.get(item, 0) for item in order.
127                     items)
128
129         # VIP discount
130         if 'VIP' in order.customer:
131             total *= 0.9
132             print(f"[PRICING] VIP discount applied")
133
134         order.price = total
135         order.status = OrderStatus.PRICED
136         print(f"[PRICING] Total: ${total:.2f}")
137         return order
138
139 class LogisticsAgent:
140     def process(self, order: Order) -> Order:
141         """Schedule delivery."""
142         print(f"\n[LOGISTICS] Scheduling delivery")
143
144         # Simple scheduling
145         order.delivery_date = "2025-12-20"
146         order.status = OrderStatus.SCHEDULED
147         print(f"[LOGISTICS] Delivery: {order.delivery_date}")
148         return order
149
150 class PaymentAgent:
151     def process(self, order: Order) -> Order:
152         """Process payment."""
153         print(f"\n[PAYMENT] Processing payment of ${order.price:.2f}
154     ")
155
156         # Simulate payment
157         order.payment_id = f"PAY-{order.order_id}"
158         order.status = OrderStatus.PAID
159         print(f"[PAYMENT] Payment successful: {order.payment_id}")
160         return order
161
162     # Run example
163     system = OrderProcessingSystem()
164
165     order1 = Order(
166         order_id="ORD-001",
167         customer="John Doe VIP",
168         items=['laptop', 'mouse']
169     )
170
171     result = system.process_order(order1)

```

```

171 print(f"\nFinal Status: {result.status.value}")
172 print(f"Price: ${result.price:.2f}")
173 print(f"Delivery: {result.delivery_date}")

```

2.8.6 דוגמה 2: סוכן מכירות + סוכן מלאי + סוכן לוגיסטיקה

דוגמה לתקשורת ישירה בין שלושה סוכנים באמצעות ארכיטקטורת Mesh.

Listing 6.7: Python: Multi-Agent Mesh Communication

```

1 from typing import Optional, Dict
2
3 class Message:
4     def __init__(self, from_agent: str, to_agent: str,
5                  msg_type: str, content: Dict):
6         self.from_agent = from_agent
7         self.to_agent = to_agent
8         self.msg_type = msg_type
9         self.content = content
10
11 class Agent:
12     def __init__(self, name: str):
13         self.name = name
14         self.inbox = []
15
16     def send_message(self, to_agent: 'Agent',
17                      msg_type: str, content: Dict):
18         """Send message to another agent."""
19         msg = Message(self.name, to_agent.name, msg_type, content)
20         to_agent.receive_message(msg)
21         print(f"[{self.name}] -> [{to_agent.name}]: {msg_type}")
22
23     def receive_message(self, msg: Message):
24         """Receive message from another agent."""
25         self.inbox.append(msg)
26
27     def process_messages(self):
28         """Process all pending messages."""
29         while self.inbox:
30             msg = self.inbox.pop(0)
31             self.handle_message(msg)
32
33     def handle_message(self, msg: Message):
34         """Handle incoming message - to be overridden."""
35         pass
36
37 class SalesAgent(Agent):
38     def __init__(self):
39         super().__init__("SalesAgent")
40         self.current_order = None
41
42     def create_order(self, items: list,

```

```

43     inventory_agent: Agent, logistics_agent: Agent)
44     :
45     """Create new order and coordinate with other agents."""
46     self.current_order = {
47         'order_id': 'ORD-123',
48         'items': items,
49         'status': 'pending'
50     }
51
52     print(f"\n[{self.name}] Creating order: {items}")
53
54     # Ask inventory about availability
55     self.send_message(
56         inventory_agent,
57         'check_availability',
58         {'items': items, 'order_id': 'ORD-123'}
59     )
60
61     def handle_message(self, msg: Message):
62         """Handle responses from other agents."""
63         if msg.msg_type == 'availability_confirmed':
64             print(f"[{self.name}] Stock confirmed!")
65             # Can now proceed with pricing, etc.
66
67         elif msg.msg_type == 'availability_denied':
68             print(f"[{self.name}] Out of stock: {msg.content}")
69             self.current_order['status'] = 'backorder'
70
71     class InventoryAgent(Agent):
72         def __init__(self):
73             super().__init__("InventoryAgent")
74             self.stock = {'laptop': 5, 'mouse': 20}
75
76         def handle_message(self, msg: Message):
77             """Handle inventory requests."""
78             if msg.msg_type == 'check_availability':
79                 items = msg.content['items']
80                 all_available = all(
81                     self.stock.get(item, 0) > 0 for item in items
82                 )
83
84                 if all_available:
85                     print(f"[{self.name}] All items available")
86                     # Respond to sales agent
87                     # (in real system would use reference to sender)
88                 else:
89                     print(f"[{self.name}] Some items unavailable")
90
91     class LogisticsAgent(Agent):
92         def __init__(self):
93             super().__init__("LogisticsAgent")

```

```

93
94     def handle_message(self, msg: Message):
95         """Handle logistics requests."""
96         if msg.msg_type == 'schedule_delivery':
97             print(f"[{self.name}] Scheduling delivery")
98
99     # Create agents
100    sales = SalesAgent()
101    inventory = InventoryAgent()
102    logistics = LogisticsAgent()
103
104    # Process order
105    sales.create_order(['laptop', 'mouse'], inventory, logistics)
106    inventory.process_messages()

```

3.8.6 דוגמה 3: מערכת `tneqA-itluM` לתמיכה ל��וחות

מערכת עם סוכני תמיכה מרובים שמתפלים בפניות שונות בו-זמנית.

מערכת תמיכה

סוכנים:

- `tneqA retuoR`: מנתב פניות לסוכן המתאים
- `tneqA troppuS lacinhceT`: תמיכה טכנית
- `tneqA gnilliB`: שאלות חיב
- `tneqA troppuS lareneG`: שאלות כלליות
- `tneqA noitalacsE`: טיפול בעיות מורכבות

זרימה:

- 1.LKoch שולח שאלה
- 2.`retuoR` מנתח את השאלה וקובע לאיזה סוכן לשלווח
- 3.הסוכן המתאים מטפל בפניה
- 4.אם הבעיה מורכבת מדי, הסוכן מעביר ל-`noitalacsE`
- 5.`tneqA noitalacsE` יכול לשתף פעולה עם מספר סוכנים או להעביר לאדם

9.6 תרגילים

1.9.6 תרגיל 1: תכנון מערכת tnegA-itluM

תכנון מערכת tnegA-itluM

תרכיש: בנק רוצה לבנות מערכת לאישור הלוואות באופן אוטומטי.

משימה:

1. זהה את הסוכנים הנדרשים (פחות 4)
2. בחר ארכיטקטורה מתאימה (lacihcraleiH / hseM / ekopS-dna-buH)
3. צייר תרשימים של הארכיטקטורה
4. הגדר את התקשרות בין הסוכנים
5. זהה קונפליקטים אפשריים והצע פתרון

פתרון מוצע:

1. סוכנים נדרשים:

□ tnegA noitacilppA: מקבל את הבקשה, מוודא שהיא מלאה

□ tnegA erocS tiderC: בודק ציון אשראי של הלקוח

□ tnegA noitacifireV emocnI: מאמת הנסיבות

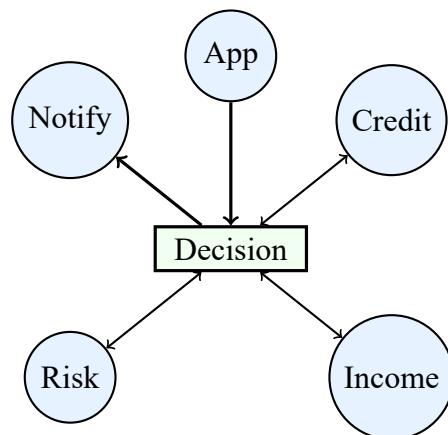
□ tnegA tnemssessA ksiR: מערך סיכון

□ tnegA noisiceD: מחליט האם לאשר או לדוחות

□ tnegA noitacifitoN: שולח הודעה ללקוח

2. ארכיטקטורה: Hub-and-Spoke. Decision Agent עם Hub-and-Spoke: החלטה צריכה להתקבל במקום אחד, על בסיס כל הנתונים. Hub-and-Spoke מבטיח שכל המידע עובר דרך סוכן החלטה אחד.

3. תרשימים: איור 9.6 מציג את הארכיטקטורה המוצעת.



איור 9.6: מערכת אישור הלוואות בארכיטקטורת Hub-and-Spoke

4. תקשורת:

□ $\text{tnegA noisiceD} \rightarrow \text{tnegA noitacilppA}$

□ $\text{ksiR/emochn/tiderC} \leftrightarrow \text{tnegA noisiceD}$

□ $\text{tnegA noitacifitoN} \rightarrow \text{tnegA noisiceD}$

5. קונפליקטים אפשריים:

□ **קונפליקט:** אומר "אשר", אבל $\text{tnemssessA ksiR erocS tiderC}$ אומר "דחה"

□ **פתרון:** משקלל את כל הגורמים לפי משקלות מוגדרים מראש (למשל: $\text{tiderC ,%03 emocnI ,%04 ksiR ,%03 noisiceD}$)

2.9.6 תרגיל 2: זיהוי נקודות כשל

זיהוי נקודות כשל

נתונה מערכת Multi-Agent בארכיטקטורת Hub-and-Spoke

□ $\text{tnegA rotartsehcrO :buH}$

□ $\text{stnega rekrow 5 :sekopS}$

שאלות:

1. מה קורה אם ה- rotartsehcrO נכשל?

2. מה קורה אם אחד מה- rekrow נכשל?

3. איך אפשר להפוך את המערכת לחסינה יותר?

פתרון:

1. אם rotartsehcrO נכשל:

□ כל המערכת נופלת -- אף rekrow לא יכול לקבל משימות

□ זהה נקודת כשל ייחודית קריטית

2. אם rekrow נכשל:

□ המערכת ממשיכה לעבוד

□ המשימות של ה- rekrow הנכשל לא יבוצעו

□ rotartsehcrO צריך להיות ולנתב מחדש

3. הפיכת המערכת לחסינה יותר:

פתרון א: $\text{ycenadnudeR rotartsehcrO}$

□ הרץ שני srotartsehcrO במצב Active-Passive

□ אם הראשי נכשל, הגיבוי נכנס לפעולה

פתרון ב: $\text{skcehC htlaeH rekrow}$

- ○ rotartsehcrO בודק כל 10 שניות שכל ה-W-Chaim srekro
 - אם W-rekro לא עונה, rotartsehcrO מפסיק לשולח לו MERCHANTABILITY
 - כשה-W-rekro חזרה, הוא מצטרף בחזרה
- פתרון ג:**
- אם W-rekro לא מחזיר תשובה תוך זמן מוגדר
 - ○ rotartsehcrO שלוח את המשימה ל-W-rekro אחר

3.9.6 תרגיל 3: בניית מדיניות noitalacsE

מדיניות noitalacsE

בנה מדיניות Escalation למערכת תמיכת ל Kohoot עם 3 רמות:

1. 1L: troppuS סוכן אוטומטי
2. 2L: troppuS מתקדם
3. 3L: troppuS אדם (מומחה)

הגדך:

- מתי עבר מ-1L ל-2L?
- מתי עבר מ-2L ל-3L?
- מהם הקритריונים?

פתרון מוצע:
מעבר מ-1L ל-2L:

1. נמוך: אם 1L לא בטוח בתשובה (7.0 > ecnedifnoc erocS ecnedifnoC).
2. מסחר ניסיונות: אם 1L ניסה 3 פעמים ולא הצליח לפטור.
3. בקשה מפורשת: אם הלוקה מבקש "לדבר עם מישחו אחר".
4. נושא מורכב: אם הנושא בקטגוריה מוגדרת כ מורכבת.

מעבר מ-2L ל-3L:

1. בעיה קריטית: תקלת המערכת שמשפיעה על Kohoot רבים.
2. ערך גבוה: בעיה PIV עם בעיה שלא נפתרה ב-2L.
3. זמן ארוך: בעיה פתואה מעל 42 שעות.
4. מגבלות טכניות: 2L לא יכול לבצע פעולה (למשל: החזר כספי מעלה סכום מסוים).

קוד לדוגמה:

Listing 6.8: Python: Escalation Policy

```

1  from dataclasses import dataclass
2  from typing import Optional
3  from datetime import datetime, timedelta
4
5  @dataclass
6  class SupportTicket:
7      ticket_id: str
8      customer: str
9      issue: str
10     level: int = 1
11     attempts: int = 0
12     created_at: datetime = None
13     is_vip: bool = False
14
15     def __post_init__(self):
16         if self.created_at is None:
17             self.created_at = datetime.now()
18
19 class EscalationPolicy:
20     def should_escalate_to_l2(self, ticket: SupportTicket,
21                               confidence: float) -> bool:
22         """Check if should escalate from L1 to L2."""
23         # Low confidence
24         if confidence < 0.7:
25             return True
26
27         # Too many attempts
28         if ticket.attempts >= 3:
29             return True
30
31         # Complex category
32         complex_keywords = ['refund', 'legal', 'security breach']
33         if any(kw in ticket.issue.lower() for kw in
complex_keywords):
34             return True
35
36         return False
37
38     def should_escalate_to_l3(self, ticket: SupportTicket) -> bool:
39         """Check if should escalate from L2 to L3."""
40         # VIP customer
41         if ticket.is_vip:
42             return True
43
44         # Open too long
45         time_open = datetime.now() - ticket.created_at
46         if time_open > timedelta(hours=24):
47             return True
48
49         # Critical issue

```

```

50 |         critical_keywords = ['system down', 'data loss', 'security']
51 |     ]
52 |     if any(kw in ticket.issue.lower() for kw in
critical_keywords):
53 |         return True
54 |
55 |     return False
56
57 | def escalate(self, ticket: SupportTicket,
58 |             confidence: Optional[float] = None):
59 |     """Escalate ticket to appropriate level."""
60 |     if ticket.level == 1:
61 |         if self.should_escalate_to_l2(ticket, confidence or
1.0):
62 |             ticket.level = 2
63 |             print(f"ESCALATED to L2: Ticket {ticket.ticket_id}")
64 |
65 |     elif ticket.level == 2:
66 |         if self.should_escalate_to_l3(ticket):
67 |             ticket.level = 3
68 |             print(f"ESCALATED to L3: Ticket {ticket.ticket_id}")
69 |
70 | # Example
71 | policy = EscalationPolicy()
72 |
73 | ticket1 = SupportTicket(
74 |     ticket_id="T001",
75 |     customer="John Doe",
76 |     issue="Need refund for order",
77 |     attempts=1
78 | )
79 | policy.escalate(ticket1, confidence=0.5)
80 | # Output: ESCALATED to L2: Ticket T001

```

תרגיל 4: תכנון ניטור

תבנון ניטור

תכנן מערכת ניטור למערכת Multi-Agent עם 01 שותכנים.

הגדר:

1. אלו מטריקות לאסוף?
2. באיזו תדירות?
3. מהם ה-Thresholds (ספרים) לאזהרות?
4. איך להציג את המידע?

פתרון:

1. מטריקות לאסוף:

ברמת טווכן בודד:

□ nwod/pu :sutatS □

detelpmoC sksaT □: מספר MERCHANTABILITY שהושלמו

:deliaF sksaT □: מספר MERCHANTABILITY שנכשלו

ycnetaL egarevA □: זמן ממוצע למשימה

:כמה MERCHANTABILITY פעילות כרגע

egasU yromeM/UPC □: ניצול משאבים

ברמת המערכת:

□ tuphguorhT latoT □: MERCHANTABILITY לשניה של כל המערכת

ycnetaL dnE-ot-dnE egarevA □: זמן מקצת לקצה

:stnegA evitcA □: כמה שותכנים פעילים

eziS eueuQ egasseM □: כמה הודעות בתווך

daehrevO noitanidrooC □: עומס תיאום כולל

2. תדירות איסוף:

daoL tnerruC ,sutatS □: (כל שנייה) scirtem emit-laeR □

tuphguorhT ,ycnetaL □: (כל 01 שניות) scirtem tsaF □

stnuoc ksaT ,yromeM/UPC □: (כל דקה) scirtem wolS □

3. sdlohserhT לאזהרות:

הקריטם	טירק	הרזהא
Agent Status	-	Down
Error Rate	> 5%	> 10%
Average Latency	> 2s	> 5s
Agent Load	> 80%	> 95%
Queue Size	> 100	> 500
Coordination Overhead	> 40%	> 60%

4. הצגת מידע:
ראשי draobhsaD

- סך סוכנים פעילים, etar rorre tuphguorht כולל :lenaP weivrevO
- טבלה עם שורה לכל סוכן, עמודות: dirG tnegA ,srorrE ,ycnetaL ,daoL ,sutatS
- גרפ המראה תקשורת בין סוכנים (עובי קו = כמות הודעות) :krowteN hparG
- רישימת התראות אקטיביות :lenaP treIA
- ycnetal :trahC enilemiT לאורך זמן

5.9.6 תרגיל 5: ניתוח עלויות מול יעילות

ניתוח עלויות

חברה שוקלת בין שתי אפשרויות:
אפשרות A: סוכן יחיד גדול

- עלות פיתוח: \$000,05
- עלות תפעול חודשית: \$1,000
- זמןعيובדמשימה: 01 שניות
- יכולת: 001 משימות/דקה

אפשרות B: 5 סוכנים קטנים

- עלות פיתוח: \$000,08 (5 * \$000,61)
- עלות תפעול חודשית: \$1,005,5
- זמןعيובדמשימה: 5 שניות (כל סוכן)
- יכולת: 002 משימות/דקה (במקביל)

שאלות:

1. מהי נקודת האיזון ?(neve-kaerb)
2. איזו אפשרות עדיפה לטוווח ארוך (3 שנים)?
3. מahan הבעיות הנוספות שצריך לחת בחשבונו?

פתרונות:

1. **הוצאות כוללות:**
אפשרות A:

$$\text{Total Cost (t months)} = 50,000 + 1,000t$$

אפשרות B:

$$\text{Total Cost (t months)} = 80,000 + 1,500t$$

נקודות איזון:

$$50,000 + 1,000t = 80,000 + 1,500t$$

$$-500t = 30,000$$

$$t = -60$$

מכיוון ש- t שלילי, אין נקודות איזון -- אפשרות A תמיד זולה יותר מבחינה כלכלית נטו!
אבל, זה לא כל הסיפור...

2. ניתוח ערך (3 שנים = 63 חודשים):

אפשרות A:

$$\text{Cost}_A = 50,000 + 1,000 \times 36 = 86,000$$

תפוקה: 001 שימושות/דקה

אפשרות B:

$$\text{Cost}_B = 80,000 + 1,500 \times 36 = 134,000$$

תפוקה: 002 שימושות/דקה

הוצאות למשימה:

נניח 1,000,000 שימושות בשנה:

אפשרות A:

$$\text{Cost per task} = \frac{86,000}{3,000,000} = \$0.0287$$

אפשרות B:

$$\text{Cost per task} = \frac{134,000}{3,000,000} = \$0.0447$$

נראה שאפשרות A זולה יותר!

אבל, מה אם הביקוש גדול?

אם מגיעות 003 שימושות/דקה:

□ **אפשרות A:** לא מספקה (רכ 001/דקה) -- צריך להוסיף סוכנים

□ **אפשרות B:** עדין מספקה (002/דקה)

אם צריך להכפיל את A:

$$\text{Cost}_A = 2 \times 86,000 = 172,000$$

הוצאות B זולה יותר!

3. עלויות נוספות:

□ **תחזוקה:** tnegA-itluM מרכיב יותר לתחזוקה (+02%)

□ **nitowr:** צריך כלים לניטור tnegA-itluM (+005\$/חודש)

□ **תקלות:** אם סוכן בודד נופל, כל המערכת נופלת (הוצאות emitnwod)

- גמישות: tnegA-itluM מאפשר שדרוגים חלקיים (חיסכון עתידי)
 - סקלබליות: tnegA-itluM קל יותר להרחבה
 - המלצת:
 - אם הביקוש צפוי ויציב (< 100 משימות/דקה): אפשרות A
 - אם הביקוש משתנה או צפוי לצמיחה: אפשרות B
 - אם חשוב זמינות גבואה (emittpu): אפשרות B

6.9.6 תרגיל 6: תקשורת בין שני סוכנים (nohtyP)

Pnohty: בסיסית תקשורת

בנה מערכת תקשורת פשוטה בין שני סוכנים:

שואל שאלת tnegA A : שואל שאלת □

B tnegA ☐ עונה על השאלה:

יש לישם:

1. מחלוקת Message

2. מחלקת Agent בסיסית

3. מגנון שליחה וקבלת

4. לוג של כל הודעה

פתרונות:

פתרונות:

Listing 6.9: Python: Basic A2A Communication

```
1 from typing import Dict, List, Optional
2 from dataclasses import dataclass, field
3 from datetime import datetime
4 import json
5
6 @dataclass
7 class Message:
8     """A2A message."""
9     msg_id: str
10    from_agent: str
11    to_agent: str
12    msg_type: str
13    payload: Dict
14    timestamp: datetime = field(default_factory=datetime.now)
15
16    def to_json(self) -> str:
17        """Convert message to JSON."""
18        return json.dumps({
19            'msg_id': self.msg_id,
```

```

20         'from': self.from_agent,
21         'to': self.to_agent,
22         'type': self.msg_type,
23         'payload': self.payload,
24         'timestamp': self.timestamp.isoformat()
25     }, indent=2)
26
27 class MessageBroker:
28     """Central message broker for A2A communication."""
29     def __init__(self):
30         self.agents: Dict[str, 'Agent'] = {}
31         self.message_log: List[Message] = []
32
33     def register_agent(self, agent: 'Agent'):
34         """Register agent with broker."""
35         self.agents[agent.name] = agent
36         print(f"[BROKER] Registered: {agent.name}")
37
38     def send_message(self, msg: Message):
39         """Route message to recipient."""
40         self.message_log.append(msg)
41
42         print(f"\n[BROKER] Routing message:")
43         print(f"  From: {msg.from_agent}")
44         print(f"  To: {msg.to_agent}")
45         print(f"  Type: {msg.msg_type}")
46
47         if msg.to_agent in self.agents:
48             self.agents[msg.to_agent].receive_message(msg)
49         else:
50             print(f"[BROKER] ERROR: Agent {msg.to_agent} not found")
51
52     def print_log(self):
53         """Print all messages."""
54         print('='*60)
55         print("MESSAGE LOG")
56         print('='*60)
57         for i, msg in enumerate(self.message_log, 1):
58             print(f"\n--- Message {i} ---")
59             print(msg.to_json())
60
61 class Agent:
62     """Basic agent with A2A capabilities."""
63     def __init__(self, name: str, broker: MessageBroker):
64         self.name = name
65         self.broker = broker
66         self.inbox: List[Message] = []
67         self.msg_counter = 0
68
69         # Register with broker

```

```

70     broker.register_agent(self)
71
72     def send(self, to_agent: str, msg_type: str, payload: Dict):
73         """Send message to another agent."""
74         self.msg_counter += 1
75         msg = Message(
76             msg_id=f'{self.name}-{self.msg_counter}',
77             from_agent=self.name,
78             to_agent=to_agent,
79             msg_type=msg_type,
80             payload=payload
81         )
82
83         print(f'\n[{self.name}] Sending: {msg_type} to {to_agent}')
84         self.broker.send_message(msg)
85
86     def receive_message(self, msg: Message):
87         """Receive message."""
88         self.inbox.append(msg)
89         print(f'[{self.name}] Received: {msg.msg_type}')
90         self.handle_message(msg)
91
92     def handle_message(self, msg: Message):
93         """Handle incoming message - to be overridden."""
94         print(f'[{self.name}] Processing: {msg.msg_type}')
95
96     class QuestionAgent(Agent):
97         """Agent that asks questions."""
98         def ask_question(self, to_agent: str, question: str):
99             """Ask a question."""
100            self.send(
101                to_agent,
102                'question',
103                {'question': question}
104            )
105
106        def handle_message(self, msg: Message):
107            """Handle answer."""
108            if msg.msg_type == 'answer':
109                answer = msg.payload.get('answer')
110                print(f'[{self.name}] Got answer: {answer}')
111
112    class AnswerAgent(Agent):
113        """Agent that answers questions."""
114        def __init__(self, name: str, broker: MessageBroker):
115            super().__init__(name, broker)
116            self.knowledge = {
117                'What is AI?': 'Artificial Intelligence',
118                'What is A2A?': 'Agent-to-Agent communication',
119                'What is Python?': 'A programming language'
120            }

```

```

121
122     def handle_message(self, msg: Message):
123         """Handle question and send answer."""
124         if msg.msg_type == 'question':
125             question = msg.payload.get('question')
126             answer = self.knowledge.get(question, 'I don\'t know')
127
128             print(f"[{self.name}] Question: {question}")
129             print(f"[{self.name}] Answer: {answer}")
130
131             # Send answer back
132             self.send(
133                 msg.from_agent,
134                 'answer',
135                 {'answer': answer, 'original_question': question}
136             )
137
138 # Example usage
139 print("*" * 60)
140 print("A2A COMMUNICATION DEMO")
141 print("*" * 60)
142
143 # Create broker
144 broker = MessageBroker()
145
146 # Create agents
147 alice = QuestionAgent("Alice", broker)
148 bob = AnswerAgent("Bob", broker)
149
150 # Alice asks questions
151 alice.ask_question("Bob", "What is AI?")
152 alice.ask_question("Bob", "What is A2A?")
153
154 # Print message log
155 broker.print_log()

```

01.6 סיכום

בפרק זה למדנו:

- **פרוטוקול A2A:** תקשורת בין סוכנים אוטונומיים
- **ארכיטקטורות:** lacihcraleiH ,hseM ,ekopS-dna-buH
- **תיאום שימוש:** חלוקה סטטית, דינמית ובסיסת יכולות
- **קונפליקטים:** זיהוי וטיפול במצב התנגשות
- **תשמור:** שימוש ב-LGparGgnaL לניהול זרימות עבודה
- **ניטור:** מטריקות וכליים למעקב אחרי MtnegA-itluM

□ **דוגמאות מעשיות:** מערכות לעיבוד הזמןות ותמיכת ל��וחות

1.01.6 נקודות מפתח למנהלים

1. **tnegA-itluM לא תמיד הפתרון:** לעיתים סוכן יחיד פשוט יותר וזול יותר
2. **תכנון הארכיטקטורה קרייטי:** בחירת ארכיטקטורה שגואה עלולה ליצור בעיות תחזוקה
3. **תיאום עולה בסוף:** יכול להיות 20%-50% מהזמן
4. **טיפול בקונפליקטים הכרחי:** צורך מנגנונים ברורים לפתרון סכunciosים
5. **ניתור הוא evah-tsum:** ללא ניתור, בלתי אפשרי לנחל tnegA-itluM ביצור
6. **התחל קטן:** עדיף להתחיל עם 2-3 סוכנים ולהוסיף בהדרגה

2.01.6 המשך

בפרק הבא נלמד על GAR (noitareneG detnemguA-laveirteR) -- טכניקה המאפשרת לסוכנים לגשת מידע עדכני ומדויק מוחץ ל-MLL, דרך אחיזור ממאג'ר ידע.

מקורות

[https://a2a-\(0.1v TFARD\) noitacificepS locotorP \(A2A\) tnegA\tnegA ,tcejorP A2A gnisu noitacificeps locotorp laiciffO ,protocol.org/latest/specification/.5202 ,\(S\)PTTH revo 0.2 CPR-NOSJ](https://a2a-(0.1v TFARD) noitacificepS locotorP (A2A) tnegA\tnegA ,tcejorP A2A gnisu noitacificeps locotorp laiciffO ,protocol.org/latest/specification/.5202 ,(S)PTTH revo 0.2 CPR-NOSJ) [1]

tnega-itluM gnivlovE nA :sisongaiD ot ataD yilibavresbO morF`` ,la te ,uiL .W ,nehC .F rof tnegAspo ,5202 ,tnirperp viXra ",smetsyS duolC ni tnemeganaM tnedicnI rof metsyS .2510.24145 :viXra .secart dna ,sgol ,scirtem gnisu tnemeganam tnedicni duolc

rof krowemarF tnegA-itluM lacihcrareiH A :artsehcrOtnegA`` ,la te ,gnaW .J ,nehC .L lartnec htiw krowemarf lacihcrareiH ,5202 ,tnirperp viXra ",gnivloS ksaT esopruP-lareneG .2506.12508 :viXra .sksat-bus setageled dna sevitcejbo sesopmoced taht tnega gninnalp

[https://developers.\(A2A\) locotorP tnegA\tnegA eht gnicnuonnA ,elgoog ,googleblog.com/en/a2a-a-new-era-of-agent-interoperability/.5202 ,ylibareporetni tnega rof locotorp A2A fo tnemecnuonna laiciffO](https://developers.(A2A) locotorP tnegA\tnegA eht gnicnuonnA ,elgoog ,googleblog.com/en/a2a-a-new-era-of-agent-interoperability/.5202 ,ylibareporetni tnega rof locotorp A2A fo tnemecnuonna laiciffO) [4]

dna ,scitylanA ,yilibavresbO :gnikramhcneB xoB-kcalB dnoyeB`` ,la te ,eeL .S ,nosnhoJ .D ni segnellahc euqinu sesserddA ,5202 ,tnirperp viXra ",smetsyS citnegA fo noitazimitpO .2503.06745 :viXra .snoitacilppa desab-MLL gnirotinom

",GAD cimanyD htiw noitanidrooC tnegA-itluM gnicnahne :CAMED`` ,la te ,kraP .J ,miK .S sesserddA ,5202 PLNME :scitsiugniL lanoitatupmoC rof noitaicossA eht fo sgnidniF ni lanoitatupmoC rof noitaicossA ,sessecorp tnega-itluM ni ycnedneped dna ytniatrecnu ksat .5202 ,scitsiugniL

<https://blog.langchain.,swolfkroW tnegA-itluM :hparGgnal ,niahCgnal tnega-itluM no ediug laiciffO ,com/langgraph-multi-agent-workflows/.4202 ,noitcurtsnoc wolfkrow> [7]

<https://www.,krowemarF noitartsehcrO tnegA leveL-woL :hparGgnal ,niahCgnal -itluM rof hparGetatS htiw noitartsehcro desab-GAD ,langchain.com/langgraph .5202 ,swolfkrow tnega> [8]

gnisU krowemarF lanoitasrevnoC tnegA-itluM A :retsaMtneGA`` ,la te ,gnahZ .K ,iL .M ,tnirperp viXra ",sisylanA dna laveirteR noitamrofnI ladomitluM rof slocotorP PCM dna A2A :viXra .noitanidrooc tnega-itluM rof slocotorp PCM dna A2A gnnibmoc krowemarF ,5202 .2507.21105

<https://,tcejorP locotorP tnegA\tnegA eht sehcnuAL noitadnuoF xuniL ,noitadnuoF xuniL //www.linuxfoundation.org/press/linux-foundation-launches-the-agent2agent-protocol-project-to-enable-secure-intelligent-noitadnuoF xuniL rednu ecnanrevog A2A ,communication-between-ai-agents .5202 ,snoitazinagro gnitropus +051 htiw> [01]

"weiveR lacinhceT :smetsyS tnegA-itluM ni seuqinhceT noituloseR tcilfnoC`` ,la te htimS .J [11]
,tcilfnoc laog otni noituloser tcilfnoc sezirogetaC ,5202 ,lanruoJ seidutS nredoM naeporuE
.gnigrem feileb dna ,tcilfnoc noitamrofni

-noisiceD evitarepooC tnegA-itluM no yevruS evisneherpmoC A`` ,la te ,gnahZ .Y ,gnaW .L [21]
noitulovE ,5202 ,tnirperp viXra ",sevitcepsreP dna segnellahC ,sehcaorppA ,soiranecS :gnikaM
.2503.13415 :viXra .gnikam-noisiced tnega-itlum xelpmoc ot tnega-elgnis morf

ni noituloseR tcilfnoC dna noitarepooC ,noitanidrooC`` ,sgnninnej .N dna egdirdlooW .M [31]
noitanidrooc SAM no krow lanoitadnuoF ,noitamotuA fo koobdnaH ni ",smetsyS tnegA-itluM
10.1007/978-1-4020-6268-1_87 :IOD .7002 ,regnirpS ,smsinahcem

",sMLL fo yevruS A :smsinahceM noitaroballoC tnegA-itluM`` ,la te ,lasnaB .G ,uW .Q [41]
,epyT ,srotca yb snoitaroballoc gniziretcarahc yevrus evisneherpmoC ,5202 ,tnirperp viXra
.2501.06322 :viXra .ygetarts dna ,erutcurts

viXra ",noitartsehcrO gnivlovE aiv noitaroballoC tnegA-itluM`` ,la te ,uiL .C ,gnahZ .W [51]
noitaroballoc tnega-itlum desab-MLL rof mgidarap elyts-reeteppup sesoporP ,5202 ,tnirperp
.2505.19591 :viXra .rotartsehcro deniart-LR htiw