Fincoder Multi-Agent Swarm Architecture | Banking App Development Assistant

Version: 1.0 | **Framework:** LangGraph StateGraph | **Domain:** Banking Application Development
Autor: João Gabriel Lima

1. DOMAIN ANALYSIS & ARCHITECTURE REFINEMENT

1.1. Real-World Scenario Analysis

Refined Domain Functionality Grouping based on actual user flows:

Grupo 1: User Session & Context Management

- User profile management e autenticação automática
- Session state restoration e continuidade
- Context preservation entre sessões
- Adaptive conversation management baseado em timing

Grupo 2: Banking App Prototyping (Hybrid ReAct + Custom Workflow)

- Temenos Digital Customization workflow
- Page selection e branding data extraction
- Interactive element identification e modification
- Sub-workflow orchestration para customização detalhada

Grupo 3: Project Scope Discovery

- Structured scope analysis workflow
- Requirements gathering e documentation
- Project planning e milestone tracking

1.2. Refined Agent Architecture

Main Agent (Session Coordinator)

- **Tipo:** ReAct Agent (conversational, adaptive timing)
- **Responsabilidades:**
  - User welcome vs returning user detection
  - System capabilities presentation
  - Intelligent routing baseado em user intent
  - Session continuity management

Prototype Agent (Hybrid Architecture)

- **Tipo:** ReAct + Custom Workflow Combination
- **Responsabilidades:**
  - Branding data collection/extraction
  - Page selection e management
  - Interactive element manipulation
  - Sub-workflow orchestration for detailed customization

Scope Discovery Agent (ReAct)

- **Tipo:** ReAct Agent (structured analysis)
- **Responsabilidades:**
  - Project scope analysis workflow
  - Requirements documentation
  - Milestone planning

2. REFINED STATE ARCHITECTURE

2.1. Enhanced Core Data Models

```python
from pydantic import BaseModel, Field
from typing import Optional, List, Literal
from datetime import datetime
from enum import Enum

# ================================
# ENHANCED CORE DATA STRUCTURES
# ================================

class UserData(BaseModel):
    """Enhanced user profile with session management"""
    # PROPÓSITO: Identificação única do usuário
    # SETADO EM: user_lookup_tool via fetch_user_profile()
    # USADO EM: user_lookup_tool, session management, handoff_tools
    # EXEMPLO: "usr_12345"
    user_id: str

    # PROPÓSITO: Contato do usuário para comunicação
    # SETADO EM: user_lookup_tool via fetch_user_profile()
    # USADO EM: Prompts de personalização, notificações
    # EXEMPLO: "joao@banco.com"
    email: str

    # PROPÓSITO: Nome para personalização de mensagens
    # SETADO EM: user_lookup_tool via fetch_user_profile()
    # USADO EM: Main agent prompt templates, welcome messages
    # EXEMPLO: "João Silva"
    name: Optional[str] = None

    # PROPÓSITO: Função do usuário para contexto de trabalho
    # SETADO EM: user_lookup_tool via fetch_user_profile()
    # USADO EM: Main agent prompts, capability filtering
    # EXEMPLO: "Product Manager"
    role: Optional[str] = None

    # PROPÓSITO: Idioma preferido para interface
    # SETADO EM: user_lookup_tool via fetch_user_profile()
    # USADO EM: Response generation, prompt templates
```

```
    # EXEMPLO: "pt-BR"
    language: str = "pt-BR"

    # PROPÓSITO: Detecta primeiro acesso para welcome flow
    # SETADO EM: user_lookup_tool baseado em historical data
    # USADO EM: conversation_continuity_tool, main agent prompts
    # IMPACTO: Determina tipo de welcome (novo vs retornando)
    # EXEMPLO: True para primeira sessão
    is_new_user: bool = True

    # PROPÓSITO: Rastreamento de última atividade
    # SETADO EM: user_lookup_tool via fetch_user_profile()
    # USADO EM: conversation_continuity_tool para timing decisions
    # EXEMPLO: "2024-01-15T10:30:00Z"
    last_login: Optional[str] = None

    # PROPÓSITO: Tempo desde última atividade para contexto conversacional
    # SETADO EM: user_lookup_tool via calculate_time_interval()
    # USADO EM: conversation_continuity_tool, main agent prompt conditionals
    # IMPACTO: Determina estilo conversacional (continue/recap/welcome back)
    # EXEMPLO: 45 (minutos)
    interval_since_latest_update: Optional[int] = None

class BankData(BaseModel):
    """Bank information and context"""
    # PROPÓSITO: Nome do banco para contexto de customização
    # SETADO EM: user_lookup_tool via fetch_bank_data()
    # USADO EM: Prompts de personalização, branding context
    # EXEMPLO: "Banco do Brasil"
    bank_name: str

    # PROPÓSITO: Website do banco para extração de branding
    # SETADO EM: user_lookup_tool via fetch_bank_data()
    # USADO EM: branding_extraction_handoff_tool, prototype agent routing
    # IMPACTO: Determina se branding pode ser extraído automaticamente
    # EXEMPLO: "https://bb.com.br"
    website: Optional[str] = None

class BrandingData(BaseModel):
    """Enhanced branding configuration with source tracking"""
    # PROPÓSITO: Cor primária do banco para customização de UI
    # SETADO EM: branding_extraction_workflow ou human_in_the_loop
    # USADO EM: page customization, HTML styling, UI generation
```

```
    # EXEMPLO: "#003366"
    primary_color: Optional[str] = None

    # PROPÓSITO: Cor secundária do banco para customização de UI
    # SETADO EM: branding_extraction_workflow ou human_in_the_loop
    # USADO EM: page customization, HTML styling, UI generation
    # EXEMPLO: "#FFD700"
    secondary_color: Optional[str] = None

    # PROPÓSITO: URL ou path do logo do banco
    # SETADO EM: branding_extraction_workflow ou human_in_the_loop
    # USADO EM: page customization, header elements, branding context
    # EXEMPLO: "https://banco.com/logo.png"
    bank_logo: Optional[str] = None

    # PROPÓSITO: Origem dos dados de branding para auditoria
    # SETADO EM: branding_extraction_workflow (EXTRACTED) ou human_in_the_loop (USER)
    # USADO EM: Quality validation, branding confidence assessment
    # EXEMPLO: "EXTRACTED"
    source: Literal["WEBSITE", "USER", "EXTRACTED"] = "USER"

    # PROPÓSITO: Completude dos dados para workflow routing
    # SETADO EM: mark_complete() method, branding validation
    # USADO EM: prototype agent routing decisions, conditional prompts
    # IMPACTO: Determina se branding extraction é necessária
    # EXEMPLO: True quando todos os campos obrigatórios estão preenchidos
    is_complete: bool = False

    def mark_complete(self) -> bool:
        """Check if all required branding data is present"""
        self.is_complete = all([
            self.primary_color,
            self.secondary_color,
            self.bank_logo
        ])
        return self.is_complete

class HTMLElement(BaseModel):
    """HTML Element representation for customization"""
    # PROPÓSITO: Identificador único do elemento HTML
    # SETADO EM: get_available_pages_tool via page loading
    # USADO EM: ui_highlight_elements_tool, get_element_by_id_tool
    # EXEMPLO: "header-logo-container"
```

```
    element_id: str

    # PROPÓSITO: Tipo de elemento HTML para customização adequada
    # SETADO EM: get_available_pages_tool via page parsing
    # USADO EM: html_change_planner_tool, customization constraints
    # EXEMPLO: "div", "button", "span"
    element_type: str

    # PROPÓSITO: Classes CSS para identificação e styling
    # SETADO EM: get_available_pages_tool via page parsing
    # USADO EM: find_elements_tool, CSS customization planning
    # EXEMPLO: ["btn", "btn-primary", "large"]
    class_names: List[str] = Field(default_factory=list)

    # PROPÓSITO: Seletor CSS para localização do elemento
    # SETADO EM: get_available_pages_tool via page parsing
    # USADO EM: ui_highlight_elements_tool, element manipulation
    # EXEMPLO: "#header .logo-container"
    css_selector: Optional[str] = None

    # PROPÓSITO: Conteúdo textual para customização
    # SETADO EM: get_available_pages_tool via page parsing
    # USADO EM: html_change_planner_tool, content modification
    # EXEMPLO: "Welcome to Your Bank"
    text_content: Optional[str] = None

class HtmlAttributes(BaseModel):
    """Atributos HTML completamente estruturados - zero dicionários"""
    style: Optional[str] = Field(description="CSS inline styles")
    css_class: Optional[str] = Field(description="CSS class names")
    data_role: Optional[str] = Field(description="Data role attribute")
    aria_label: Optional[str] = Field(description="Accessibility label")
    aria_hidden: Optional[bool] = Field(description="Hide from screen readers")
    tabindex: Optional[int] = Field(description="Tab order index")
    title: Optional[str] = Field(description="Tooltip text")
    id: Optional[str] = Field(description="Element ID")
    hidden: Optional[bool] = Field(description="Hidden attribute")
    draggable: Optional[bool] = Field(description="Draggable attribute")
    contenteditable: Optional[bool] = Field(description="Content editable flag")

class StyleChange(BaseModel):
    """Mudança específica de CSS/estilo"""
    change_type: Literal["style"] = "style"
```

```
    css_property: str = Field(description="Propriedade CSS modificada")
    old_value: Optional[str] = Field(description="Valor CSS anterior")
    new_value: str = Field(description="Novo valor CSS")

class ContentChange(BaseModel):
    """Mudança específica de conteúdo"""
    change_type: Literal["content"] = "content"
    content_type: Literal["text", "html", "markdown"] = Field(description="Tipo de cont
    old_value: Optional[str] = Field(description="Conteúdo anterior")
    new_value: str = Field(description="Novo conteúdo")
    preserve_children: bool = Field(default=True, description="Preservar elementos filh

class VisibilityChange(BaseModel):
    """Mudança específica de visibilidade"""
    change_type: Literal["visibility"] = "visibility"
    visibility_method: Literal["display", "visibility", "opacity"] = Field(description=
    old_visible: Optional[bool] = Field(description="Estado anterior")
    new_visible: bool = Field(description="Novo estado de visibilidade")

    # Union type para mudanças específicas
    HtmlChangeUnion = Union[StyleChange, ContentChange, VisibilityChange]

    # PROPÓSITO: Atributos HTML para customização avançada
    # SETADO EM: get_available_pages_tool via page parsing
    # USADO EM: html_change_planner_tool, attribute modification
    attributes: Optional[HtmlAttributes] = None

    # PROPÓSITO: Flag para controlar se elemento pode ser customizado
    # SETADO EM: get_available_pages_tool baseado em business rules
    # USADO EM: get_customizable_elements() method, UI filtering
    # IMPACTO: Elementos não customizáveis são ignorados nos workflows
    # EXEMPLO: True para elementos de branding, False para estruturais
    is_customizable: bool = True

class Question(BaseModel):
    """Predefined question structure for pages"""
    # PROPÓSITO: Identificador único da pergunta
    # SETADO EM: get_available_pages_tool via page configuration
    # USADO EM: human_in_the_loop, question tracking, UI generation
    # EXEMPLO: "q1_primary_color"
    question_id: str

    # PROPÓSITO: Texto da pergunta para o usuário
```

```python
    # SETADO EM: get_available_pages_tool via page configuration
    # USADO EM: human_in_the_loop prompts, UI question display
    # EXEMPLO: "What is your primary brand color?"
    question_text: str

    # PROPÓSITO: Tipo de input esperado para validação
    # SETADO EM: get_available_pages_tool via page configuration
    # USADO EM: human_in_the_loop validation, UI input type
    # EXEMPLO: "color" para color picker, "text" para texto livre
    question_type: Literal["text", "color", "url", "boolean", "choice"] = "text"

    # PROPÓSITO: Flag para marcar pergunta obrigatória
    # SETADO EM: get_available_pages_tool via page configuration
    # USADO EM: get_unanswered_questions(), workflow validation
    # IMPACTO: Perguntas obrigatórias bloqueiam progresso se não respondidas
    # EXEMPLO: True para dados essenciais
    required: bool = True

    # PROPÓSITO: Opções para perguntas tipo choice
    # SETADO EM: get_available_pages_tool via page configuration
    # USADO EM: human_in_the_loop quando question_type="choice"
    # EXEMPLO: ["Option A", "Option B", "Option C"]
    choices: Optional[List[str]] = None

    # PROPÓSITO: Flag para tracking de resposta
    # SETADO EM: human_in_the_loop após resposta do usuário
    # USADO EM: get_unanswered_questions(), workflow progression
    # IMPACTO: Determina se pergunta ainda precisa ser feita
    # EXEMPLO: True após usuário responder
    answered: bool = False

    # PROPÓSITO: Resposta fornecida pelo usuário
    # SETADO EM: human_in_the_loop após input do usuário
    # USADO EM: page customization, branding data, UI generation
    # EXEMPLO: "#FF5733" para question_type="color"
    answer: Optional[str] = None

class PageChanges(BaseModel):
    """Track changes made to page elements"""
    # PROPÓSITO: Identificador único da mudança para auditoria
    # SETADO EM: html_change_planner_tool, page_customization_workflow
    # USADO EM: apply_element_changes, change tracking, rollback
    # EXEMPLO: "change_001_logo_color"
```

```
    change_id: str

    # PROPÓSITO: ID do elemento HTML a ser modificado
    # SETADO EM: html_change_planner_tool baseado em user request
    # USADO EM: apply_element_changes para localizar elemento
    # EXEMPLO: "header-logo-container"
    element_id: str

    # ESTRUTURADO: Union type baseado em discriminated union
    # Cada tipo de mudança tem estrutura específica adequada
    change_data: HtmlChangeUnion = Field(discriminator='change_type')

    # PROPÓSITO: Timestamp da mudança para auditoria
    # SETADO EM: html_change_planner_tool ao criar mudança
    # USADO EM: Change history, auditing, temporal tracking
    # EXEMPLO: "2024-01-15T10:30:00Z"
    timestamp: str

    # PROPÓSITO: Flag para tracking se mudança foi aplicada
    # SETADO EM: apply_element_changes após aplicação
    # USADO EM: Workflow progression, change validation
    # IMPACTO: Determina se mudança ainda precisa ser aplicada
    # EXEMPLO: True após aplicação bem-sucedida
    applied: bool = False

    # NOVA CONEXÃO COM WORKFLOW PARA RASTREABILIDADE
    # PROPÓSITO: ID do CustomizationStep que gerou esta mudança
    # SETADO EM: execute_customization_steps durante planning
    # USADO EM: Auditoria, debugging, rollback, progress tracking
    # EXEMPLO: "step_001_change_logo"
    step_id: str = Field(description="ID do CustomizationStep que gerou esta mudança")

    # PROPÓSITO: Descrição do step para contexto
    # SETADO EM: execute_customization_steps copiado de CustomizationStep.description
    # USADO EM: User feedback, audit trails, debugging context
    # EXEMPLO: "Change primary logo color to match brand"
    step_description: str = Field(description="Descrição do step para contexto")

    # PROPÓSITO: Ordem de execução dentro do step
    # SETADO EM: execute_customization_steps durante sequenciamento
    # USADO EM: Rollback em ordem correta, dependency tracking
    # EXEMPLO: 1, 2, 3 (primeira, segunda, terceira mudança do step)
    execution_order: int = Field(description="Ordem de execução dentro do step")
```

```
    # PROPÓSITO: Status detalhado da aplicação
    # SETADO EM: apply_element_changes baseado no resultado
    # USADO EM: Error handling, retry logic, user feedback
    # EXEMPLO: "applied" se sucesso, "failed" se erro
    application_status: Literal["pending", "applied", "failed", "rolled_back"] = "pendi

    # PROPÓSITO: Mensagem de erro se aplicação falhou
    # SETADO EM: apply_element_changes durante exception handling
    # USADO EM: Debugging, user feedback, retry decisions
    # EXEMPLO: "Element not found: header-logo-container"
    error_message: Optional[str] = Field(description="Erro se aplicação falhou")

    # PROPÓSITO: Dados para rollback se necessário
    # SETADO EM: apply_element_changes antes da modificação
    # USADO EM: Rollback operations, undo functionality
    # EXEMPLO: JSON com estado anterior completo do elemento
    rollback_data: Optional[str] = Field(description="Dados para rollback se necessário

class PageModel(BaseModel):
    """Complete page model for customization"""
    # PROPÓSITO: Identificador único da página
    # SETADO EM: get_available_pages_tool via fetch_available_pages()
    # USADO EM: Page selection, tracking, UI references
    # EXEMPLO: "temenos_dashboard_v2"
    page_id: str

    # PROPÓSITO: Nome humanizado da página para apresentação
    # SETADO EM: get_available_pages_tool via fetch_available_pages()
    # USADO EM: UI display, user selection options, prompts
    # EXEMPLO: "Dashboard Principal"
    name: str

    # PROPÓSITO: Microapp ao qual a página pertence
    # SETADO EM: get_available_pages_tool via fetch_available_pages()
    # USADO EM: Categorization, workflow context, capabilities
    # EXEMPLO: "Banking Dashboard"
    microapp: str

    # PROPÓSITO: Versão da página para controle de compatibilidade
    # SETADO EM: get_available_pages_tool via fetch_available_pages()
    # USADO EM: Version control, compatibility checks
    # EXEMPLO: "2.1.0"
```

```
    version: str

    # PROPÓSITO: HTML raw da página para parsing e manipulation
    # SETADO EM: get_available_pages_tool via fetch_available_pages()
    # USADO EM: Element extraction, page rendering, customization base
    # EXEMPLO: "<html><head>...</head><body>...</body></html>"
    html_raw: str

    # PROPÓSITO: Lista de elementos customizáveis da página
    # SETADO EM: get_available_pages_tool via page parsing
    # USADO EM: ui_highlight_elements_tool, get_customizable_elements()
    # EXEMPLO: [HTMLElement(...), HTMLElement(...)]
    html_elements: List[HTMLElement] = Field(default_factory=list)

    # PROPÓSITO: Timestamp da última atualização da página
    # SETADO EM: get_available_pages_tool via fetch_available_pages()
    # USADO EM: Version tracking, cache invalidation
    # EXEMPLO: "2024-01-15T10:30:00Z"
    updated_at: str

    # PROPÓSITO: Texto introdutório da página para contexto
    # SETADO EM: get_available_pages_tool via page configuration
    # USADO EM: User guidance, page presentation, context setting
    # EXEMPLO: "This dashboard provides main banking overview"
    introduction: str

    # PROPÓSITO: Perguntas predefinidas para coleta de dados
    # SETADO EM: get_available_pages_tool via page configuration
    # USADO EM: get_unanswered_questions(), human_in_the_loop
    # EXEMPLO: [Question(...), Question(...)]
    predefined_questions: List[Question] = Field(default_factory=list)

    # PROPÓSITO: Histórico de mudanças aplicadas à página
    # SETADO EM: page_customization_workflow, apply_element_changes
    # USADO EM: Change tracking, rollback, audit trail
    # EXEMPLO: [PageChanges(...), PageChanges(...)]
    changes: List[PageChanges] = Field(default_factory=list)

    def get_unanswered_questions(self) -> List[Question]:
        """Get questions that haven't been answered yet"""
        return [q for q in self.predefined_questions if not q.answered]

    def get_customizable_elements(self) -> List[HTMLElement]:
```

```
        """Get elements that can be customized"""
        return [e for e in self.html_elements if e.is_customizable]

class SystemCapability(BaseModel):
    """System capability definition"""
    # PROPÓSITO: Identificador único da capacidade do sistema
    # SETADO EM: system_capabilities_tool via get_system_capabilities()
    # USADO EM: Capability selection, UI display, routing decisions
    # EXEMPLO: "temenos_customization"
    capability_id: str

    # PROPÓSITO: Nome humanizado da capacidade
    # SETADO EM: system_capabilities_tool via get_system_capabilities()
    # USADO EM: UI display, user selection options, main agent prompts
    # EXEMPLO: "Temenos Digital Customization"
    name: str

    # PROPÓSITO: Descrição da capacidade para contexto do usuário
    # SETADO EM: system_capabilities_tool via get_system_capabilities()
    # USADO EM: User guidance, capability presentation, help text
    # EXEMPLO: "Customize Temenos Digital Banking pages and workflows"
    description: str

    # PROPÓSITO: Flag de disponibilidade da capacidade
    # SETADO EM: system_capabilities_tool via business rules
    # USADO EM: Capability filtering, UI availability, routing decisions
    # IMPACTO: Capacidades indisponíveis são ocultadas ou desabilitadas
    # EXEMPLO: True para capacidades ativas
    is_available: bool = True

    # PROPÓSITO: Agente responsável por esta capacidade
    # SETADO EM: system_capabilities_tool via configuration
    # USADO EM: Handoff routing, capability-to-agent mapping
    # EXEMPLO: "Prototype_Agent"
    entry_agent: str

class CustomizationStep(BaseModel):
    """Step in the customization process"""
    # PROPÓSITO: Identificador único do passo de customização
    # SETADO EM: html_change_planner_tool baseado em user request
    # USADO EM: Step tracking, execution order, workflow progression
    # EXEMPLO: "step_001_change_logo"
    step_id: str
```

```
    # PROPÓSITO: Descrição humanizada do passo
    # SETADO EM: html_change_planner_tool baseado em analysis
    # USADO EM: User feedback, progress display, workflow documentation
    # EXEMPLO: "Change primary logo color to match brand"
    description: str

    # PROPÓSITO: Ferramentas necessárias para executar o passo
    # SETADO EM: html_change_planner_tool baseado em step requirements
    # USADO EM: page_customization_workflow, tool validation
    # EXEMPLO: ["get_element_by_id_tool", "apply_style_changes"]
    tools_required: List[str] = Field(default_factory=list)

    # PROPÓSITO: Resultado esperado após execução do passo
    # SETADO EM: html_change_planner_tool baseado em user goals
    # USADO EM: Validation, quality checks, user confirmation
    # EXEMPLO: "Logo displays in new brand color"
    expected_outcome: str

    # PROPÓSITO: Flag de conclusão do passo
    # SETADO EM: page_customization_workflow após execution
    # USADO EM: Progress tracking, workflow continuation, validation
    # IMPACTO: Determina se passo ainda precisa ser executado
    # EXEMPLO: True após aplicação bem-sucedida
    completed: bool = False

# ================================
# ENHANCED DOMAIN STATE SCHEMAS
# ================================

class PrototypeStateSchema(BaseModel):
    """Specialized state for prototype agent workflows"""

    # PROPÓSITO: Página atualmente selecionada para customização
    # SETADO EM: human_in_the_loop após user selection, get_available_pages_tool
    # USADO EM: page_customization_workflow, HTML manipulation, UI context
    # IMPACTO: Sem página selecionada, customization workflows não podem executar
    # EXEMPLO: PageModel object with page details
    selected_page: Optional[PageModel] = None

    # PROPÓSITO: Lista de páginas disponíveis para seleção
    # SETADO EM: get_available_pages_tool via fetch_available_pages()
    # USADO EM: User selection UI, prototype agent prompts, page listing
```

```
    # EXEMPLO: [PageModel(...), PageModel(...)]
    available_pages: List[PageModel] = Field(default_factory=list)

    # PROPÓSITO: Passos do plano de customização
    # SETADO EM: html_change_planner_tool baseado em user request
    # USADO EM: page_customization_workflow, progress tracking, execution order
    # EXEMPLO: [CustomizationStep(...), CustomizationStep(...)]
    steps: List[CustomizationStep] = Field(default_factory=list)

    # PROPÓSITO: ID do passo atualmente em execução
    # SETADO EM: page_customization_workflow durante execution
    # USADO EM: Progress tracking, step navigation, workflow control
    # EXEMPLO: "step_002_change_colors"
    current_step_id: Optional[str] = None

    # PROPÓSITO: IDs dos elementos destacados na UI
    # SETADO EM: ui_highlight_elements_tool após highlighting
    # USADO EM: Visual feedback, user guidance, element selection
    # EXEMPLO: ["header-logo", "main-banner", "footer-links"]
    highlighted_elements: List[str] = Field(default_factory=list)

    # PROPÓSITO: Mudanças planejadas mas ainda não aplicadas
    # SETADO EM: html_change_planner_tool durante planning
    # USADO EM: page_customization_workflow, user confirmation, change preview
    # EXEMPLO: [PageChanges(...), PageChanges(...)]
    planned_changes: List[PageChanges] = Field(default_factory=list)

    # PROPÓSITO: Mudanças já aplicadas à página
    # SETADO EM: page_customization_workflow após application
    # USADO EM: Change history, rollback operations, audit trail
    # EXEMPLO: [PageChanges(...), PageChanges(...)]
    applied_changes: List[PageChanges] = Field(default_factory=list)

    # PROPÓSITO: Flag indicando se sub-workflow está ativo
    # SETADO EM: page_customization_handoff_tool, branding_extraction_handoff_tool
    # USADO EM: route_prototype_workflow(), workflow routing decisions
    # IMPACTO: Determina se controle deve ir para sub-workflow específico
    # EXEMPLO: True quando page customization em execução
    sub_workflow_active: bool = False

    # PROPÓSITO: Tipo de sub-workflow atualmente ativo
    # SETADO EM: handoff tools baseado em workflow type
    # USADO EM: route_prototype_workflow(), workflow routing
```

```
    # EXEMPLO: "page_customization" ou "branding_extraction"
    sub_workflow_type: Optional[Literal["page_customization", "branding_extraction"]] =

    # PROPÓSITO: Mensagens do contexto do prototype workflow
    # SETADO EM: prototype agent interactions, sub-workflow communication
    # USADO EM: Workflow communication, context preservation
    # EXEMPLO: [{"role": "user", "content": "Change logo color"}]
    messages: List[dict] = Field(default_factory=list)

    # PROPÓSITO: Flag indicando se aguarda input do usuário
    # SETADO EM: human_in_the_loop, UI interaction tools
    # USADO EM: Workflow control, UI state management, interaction flow
    # IMPACTO: Controla se workflow deve pausar aguardando user action
    # EXEMPLO: True quando aguardando page selection
    awaiting_user_input: bool = False

    # PROPÓSITO: Tipo de input esperado do usuário
    # SETADO EM: Tools baseado em interaction context
    # USADO EM: UI rendering, input validation, user guidance
    # EXEMPLO: "page_selection" para escolha de página
    input_type: Optional[Literal["page_selection", "element_selection", "confirmation"]

class FincoderSwarmState(SwarmState):
    """Enhanced state for banking app development domain"""

    # PROPÓSITO: Dados do usuário para personalização e contexto
    # SETADO EM: user_lookup_tool via fetch_user_profile()
    # USADO EM: Main agent prompts, session management, personalization
    # EXEMPLO: UserData object with user profile information
    user_data: Optional[UserData] = None

    # PROPÓSITO: Dados do banco para contexto de customização
    # SETADO EM: user_lookup_tool via fetch_bank_data()
    # USADO EM: Branding context, customization decisions, prompts
    # EXEMPLO: BankData object with bank information
    bank_data: Optional[BankData] = None

    # PROPÓSITO: Capacidades disponíveis do sistema
    # SETADO EM: system_capabilities_tool via get_system_capabilities()
    # USADO EM: Main agent capability presentation, routing decisions
    # EXEMPLO: [SystemCapability(...), SystemCapability(...)]
    system_capabilities: List[SystemCapability] = Field(default_factory=list)
```

```python
    # PROPÓSITO: Capacidade selecionada pelo usuário
    # SETADO EM: Main agent após user selection
    # USADO EM: Routing para agent especializado, workflow context
    # EXEMPLO: "temenos_customization"
    selected_capability: Optional[str] = None

    # PROPÓSITO: Descrição da última tarefa para continuidade
    # SETADO EM: user_lookup_tool via fetch_last_task_description()
    # USADO EM: conversation_continuity_tool, context restoration
    # EXEMPLO: "User was customizing dashboard colors"
    last_task_description: Optional[str] = None

    # PROPÓSITO: Resumo da última ação para contexto
    # SETADO EM: Workflow completion, action summaries
    # USADO EM: conversation_continuity_tool, session context
    # EXEMPLO: "Applied 3 color changes to dashboard page"
    last_action_summary: Optional[str] = None

    # PROPÓSITO: Dados de branding para customização
    # SETADO EM: branding_extraction_workflow, human_in_the_loop
    # USADO EM: Page customization, UI styling, prototype workflows
    # EXEMPLO: BrandingData object with colors and logo
    branding_data: BrandingData = Field(default_factory=BrandingData)

    # PROPÓSITO: Estado especializado do prototype agent
    # SETADO EM: Prototype agent workflows, sub-workflow transitions
    # USADO EM: Prototype agent routing, page customization, element manipulation
    # EXEMPLO: PrototypeStateSchema object with workflow state
    prototype: PrototypeStateSchema = Field(default_factory=PrototypeStateSchema)

    # PROPÓSITO: Status e progresso do workflow atual
    # SETADO EM: Workflow transitions, progress updates
    # USADO EM: Progress tracking, UI status, workflow control
    # EXEMPLO: WorkflowStatus object with current phase and progress
    workflow_status: WorkflowStatus = Field(default_factory=WorkflowStatus)

    # PROPÓSITO: Metadados de execução e processamento
    # SETADO EM: Tool executions, error handling, quality control
    # USADO EM: Error recovery, retry logic, execution tracking
    # EXEMPLO: ProcessingMetadata object with execution context
    processing_metadata: ProcessingMetadata = Field(default_factory=ProcessingMetadata)

class WorkflowStatus(BaseModel):
```

```python
    """Enhanced workflow status tracking"""
    # PROPÓSITO: Fase atual do workflow principal
    # SETADO EM: Workflow transitions, phase changes nos agentes
    # USADO EM: Progress tracking, UI status, prompt conditionals
    # EXEMPLO: "page_selection" quando usuário escolhe página
    current_phase: Literal["welcome", "capability_selection", "branding_setup", "page_s

    # PROPÓSITO: Sub-fase dentro da fase atual
    # SETADO EM: Sub-workflow transitions, detailed progress tracking
    # USADO EM: Detailed progress display, workflow navigation
    # EXEMPLO: "element_selection" dentro de "customization"
    sub_phase: Optional[str] = None

    # PROPÓSITO: Última ação executada no workflow
    # SETADO EM: Tool executions, agent actions
    # USADO EM: Progress history, debugging, user feedback
    # EXEMPLO: "page_customization_completed"
    last_action: Optional[str] = None

    # PROPÓSITO: Porcentagem de progresso do workflow
    # SETADO EM: Progress updates durante workflow execution
    # USADO EM: UI progress bars, completion estimation
    # EXEMPLO: 75.0 para 75% completo
    progress_percentage: float = 0.0

    # PROPÓSITO: Próxima ação recomendada ao usuário
    # SETADO EM: Workflow analysis, next step prediction
    # USADO EM: User guidance, next step suggestions
    # EXEMPLO: "select_elements_to_customize"
    next_recommended_action: Optional[str] = None

class ProcessingMetadata(BaseModel):
    """Enhanced processing context and execution data"""
    # PROPÓSITO: Step atual sendo executado
    # SETADO EM: Tool executions, workflow step transitions
    # USADO EM: Debugging, progress tracking, error context
    # EXEMPLO: "handoff_to_page_customization"
    current_step: Optional[str] = None

    # PROPÓSITO: Contador de tentativas para retry logic
    # SETADO EM: Error handling, retry mechanisms
    # USADO EM: Retry decision logic, max retry validation
    # EXEMPLO: 2 (segunda tentativa)
```

```python
    retry_count: int = 0

    # PROPÓSITO: Limite máximo de tentativas
    # SETADO EM: Initialization, configuration
    # USADO EM: Retry decision logic, error handling
    # EXEMPLO: 3 (máximo 3 tentativas)
    max_retries: int = 3

    # PROPÓSITO: Score de qualidade da execução
    # SETADO EM: Quality validation processes
    # USADO EM: Quality control decisions, retry logic
    # EXEMPLO: 0.85 (85% quality score)
    quality_score: Optional[float] = None

    # PROPÓSITO: Erros de validação encontrados
    # SETADO EM: Validation processes, error detection
    # USADO EM: Error reporting, debugging, retry decisions
    # EXEMPLO: ["Invalid color format", "Missing element ID"]
    validation_errors: List[str] = Field(default_factory=list)

    # PROPÓSITO: Contexto de execução para tracking
    # SETADO EM: Tool executions, workflow context setting
    # USADO EM: Debugging, execution analysis, context restoration
    # EXEMPLO: "page_customization_workflow"
    execution_context: Optional[str] = None

    # PROPÓSITO: Timestamp da última atualização
    # SETADO EM: Tool executions, state updates
    # USADO EM: Temporal tracking, debugging, audit
    # EXEMPLO: "2024-01-15T10:30:00Z"
    timestamp: Optional[str] = None
```

2.2. Specialized Sub-Workflow States

```python
# ================================
# SUB-WORKFLOW STATES
# ================================

class PageCustomizationState(BaseModel):
    """State for page customization sub-workflow"""

    # PROPÓSITO: Passos de customização herdados do parent workflow
    # SETADO EM: page_customization_entry_node via state conversion
    # USADO EM: execute_customization_steps, step progression
    # EXEMPLO: [CustomizationStep(...), CustomizationStep(...)]
    steps: List[CustomizationStep] = Field(default_factory=list)

    # PROPÓSITO: Mensagens do contexto do parent workflow
    # SETADO EM: page_customization_entry_node via state conversion
    # USADO EM: Context preservation, communication history
    # EXEMPLO: [{"role": "user", "content": "Change colors"}]
    messages: List[dict] = Field(default_factory=list)

    # PROPÓSITO: Página selecionada para customização
    # SETADO EM: page_customization_entry_node via state conversion
    # USADO EM: Element access, HTML manipulation, change application
    # EXEMPLO: PageModel object with page details
    selected_page: PageModel

    # PROPÓSITO: Índice do step atual sendo executado
    # SETADO EM: execute_customization_steps durante progression
    # USADO EM: Step navigation, progress tracking, completion check
    # EXEMPLO: 2 (executando terceiro step)
    current_step_index: int = 0

    # PROPÓSITO: Flag indicando se processamento iniciou
    # SETADO EM: page_customization_entry_node durante initialization
    # USADO EM: Processing state tracking, workflow control
    # EXEMPLO: True após início do processamento
    processing_started: bool = False

    # PROPÓSITO: Mudanças planejadas para aplicação
    # SETADO EM: execute_customization_steps durante planning
    # USADO EM: apply_element_changes, change preview
    # EXEMPLO: [PageChanges(...), PageChanges(...)]
```

```
    change_plan: List[PageChanges] = Field(default_factory=list)

    # PROPÓSITO: Mudanças aplicadas com sucesso
    # SETADO EM: apply_element_changes após application
    # USADO EM: page_customization_exit_node, result reporting
    # EXEMPLO: [PageChanges(...), PageChanges(...)]
    applied_changes: List[PageChanges] = Field(default_factory=list)

    # PROPÓSITO: Resumo final do resultado da customização
    # SETADO EM: finalize_customization node
    # USADO EM: page_customization_exit_node, user feedback
    # EXEMPLO: "Successfully applied 3 color changes to dashboard"
    final_thoughts: str = ""

class BrandingExtractionState(BaseModel):
    """State for branding data extraction sub-workflow"""

    # PROPÓSITO: URL do website para extração de branding
    # SETADO EM: branding_extraction_entry_node via bank_data.website
    # USADO EM: Web scraping, branding extraction process
    # EXEMPLO: "https://banco.com.br"
    website_url: str

    # PROPÓSITO: Flag indicando se extração foi iniciada
    # SETADO EM: branding_extraction_entry_node durante initialization
    # USADO EM: Process control, workflow state tracking
    # EXEMPLO: True após início da extração
    extraction_started: bool = False

    # PROPÓSITO: Cores extraídas do website
    # SETADO EM: branding_extraction_workflow durante processing
    # USADO EM: BrandingData creation, color selection
    # EXEMPLO: ["#003366", "#FFD700", "#FFFFFF"]
    extracted_colors: List[str] = Field(default_factory=list)

    # PROPÓSITO: URLs de logos encontrados no website
    # SETADO EM: branding_extraction_workflow durante processing
    # USADO EM: BrandingData creation, logo selection
    # EXEMPLO: ["https://banco.com.br/logo.png", "https://banco.com.br/favicon.ico"]
    extracted_logos: List[str] = Field(default_factory=list)

    # PROPÓSITO: Dados de branding estruturados extraídos
    # SETADO EM: branding_extraction_workflow após processing
```

```
    # USADO EM: branding_extraction_exit_node, state update
    # EXEMPLO: BrandingData object with extracted information
    extracted_branding: Optional[BrandingData] = None

    # PROPÓSITO: Flag de sucesso da extração
    # SETADO EM: branding_extraction_workflow após completion
    # USADO EM: Workflow routing, error handling, result validation
    # IMPACTO: Determina se branding data pode ser usado
    # EXEMPLO: True se extração foi bem-sucedida
    extraction_successful: bool = False

    # PROPÓSITO: Erros encontrados durante extração
    # SETADO EM: branding_extraction_workflow durante error handling
    # USADO EM: Error reporting, debugging, fallback decisions
    # EXEMPLO: ["Website inaccessible", "No valid colors found"]
    extraction_errors: List[str] = Field(default_factory=list)
```

3. REFINED AGENT SPECIFICATIONS

3.1. Main Agent (Session Coordinator) - Enhanced

Enhanced Agent Configuration:

```python
def build_main_agent(model, handoff_tools=None):
    return AgentBuilder(
        name="Session_Coordinator",
        model=model,
        tools=[
            user_lookup_tool,
            system_capabilities_tool,
            session_management_tool,
            conversation_continuity_tool,
            human_in_the_loop
        ] + (handoff_tools or []),
        agent_identity="Fincoder Session Coordinator - Temenos Digital Banking Assistan
        responsibilities=[
            "Manage user sessions and context continuity",
            "Provide adaptive conversation based on user timing",
            "Present system capabilities and route to specialists",
            "Maintain project context across sessions"
        ],
        state_schema=FincoderSwarmState,
        prompt_template_path="prompts/fincoder/main_agent.jinja2",
        dynamic_block_template_path="prompts/base_agent_prompt.jinja2",
        additional_pre_hooks=handoff_tools,
    ).build()
```

Enhanced Core Tools:

```python
def user_lookup_tool(
    user_id: str,
    tool_call_id: Annotated[str, InjectedToolCallId] = None,
) -> Command:
    """Enhanced user profile retrieval with session management"""
    profile_data = fetch_user_profile(user_id)
    bank_data = fetch_bank_data(user_id) if profile_data else None

    # Calculate session timing
    last_update = profile_data.get("last_activity")
    interval_minutes = calculate_time_interval(last_update) if last_update else None

    user_data = UserData(
        user_id=user_id,
        email=profile_data.get("email"),
        name=profile_data.get("name"),
        role=profile_data.get("role"),
        language=profile_data.get("language", "pt-BR"),
        is_new_user=profile_data.get("is_new_user", True),
        last_login=profile_data.get("last_login"),
        interval_since_latest_update=interval_minutes
    )

    bank_data_obj = BankData(**bank_data) if bank_data else None

    # Retrieve last task context
    last_task = fetch_last_task_description(user_id)

    return Command(
        update={
            "user_data": user_data,
            "bank_data": bank_data_obj,
            "last_task_description": last_task,
            "messages": [
                ToolMessage(
                    content=f"Session loaded for {user_data.name}",
                    tool_call_id=tool_call_id,
                )
            ],
        }
    )

def system_capabilities_tool(
```

```
    tool_call_id: Annotated[str, InjectedToolCallId] = None,
) -> Command:
    """Retrieve and present system capabilities"""
    capabilities_data = get_system_capabilities()

    capabilities = [
        SystemCapability(
            capability_id=cap["id"],
            name=cap["name"],
            description=cap["description"],
            is_available=cap.get("available", True),
            entry_agent=cap.get("entry_agent", "Prototype_Agent")
        )
        for cap in capabilities_data
    ]

    return Command(
        update={
            "system_capabilities": capabilities,
            "workflow_status": WorkflowStatus(
                current_phase="capability_selection",
                progress_percentage=10.0,
                next_recommended_action="select_capability"
            ),
            "messages": [
                ToolMessage(
                    content=f"Retrieved {len(capabilities)} system capabilities",
                    tool_call_id=tool_call_id,
                )
            ],
        }
    )

def conversation_continuity_tool(
    action: Literal["generate_welcome", "generate_continuation", "update_context"],
    context_data: Optional[dict] = None,
    tool_call_id: Annotated[str, InjectedToolCallId] = None,
) -> Command:
    """Manage conversation continuity based on session timing"""

    if action == "generate_welcome":
        message_type = "new_user_welcome"
    elif action == "generate_continuation":
```

```
        message_type = "returning_user_continuation"
    else:
        message_type = "context_update"

    return Command(
        update={
            "processing_metadata": ProcessingMetadata(
                current_step=action,
                execution_context=message_type,
                timestamp=datetime.now().isoformat()
            ),
            "messages": [
                ToolMessage(
                    content=f"Conversation continuity managed: {action}",
                    tool_call_id=tool_call_id,
                )
            ],
        }
    )
```

3.2. Prototype Agent (Hybrid Architecture) - Major Refinement

Hybrid Agent Architecture:

```python
def build_prototype_agent(model, handoff_tools=None):
    """Build hybrid ReAct + Custom Workflow prototype agent"""

    # Build ReAct component
    react_agent = AgentBuilder(
        name="Prototype_Agent_ReAct",
        model=model,
        tools=[
            get_available_pages_tool,
            get_element_by_id_tool,
            get_screenshot_tool,
            find_elements_tool,
            ui_highlight_elements_tool,
            html_change_planner_tool,
            page_customization_handoff_tool,
            branding_extraction_handoff_tool,
            human_in_the_loop
        ],
        agent_identity="Temenos Digital Customization Specialist",
        responsibilities=[
            "Manage page selection and branding setup",
            "Coordinate interactive element identification",
            "Plan and orchestrate customization workflows",
            "Handle user interaction and guidance"
        ],
        state_schema=FincoderSwarmState,
        prompt_template_path="prompts/fincoder/prototype_react.jinja2",
        dynamic_block_template_path="prompts/base_agent_prompt.jinja2",
        additional_pre_hooks=handoff_tools,
    ).build()

    # Build custom workflows
    page_customization_workflow = build_page_customization_workflow()
    branding_extraction_workflow = build_branding_extraction_workflow()

    # Create hybrid workflow
    hybrid_workflow = StateGraph(
        state_schema=FincoderSwarmState,
        input_schema=FincoderSwarmState,
        output_schema=FincoderSwarmState,
    )

    # Add components
```

```
    hybrid_workflow.add_node("react_agent", react_agent)
    hybrid_workflow.add_node("page_customization", page_customization_workflow)
    hybrid_workflow.add_node("branding_extraction", branding_extraction_workflow)

    # Configure routing
    hybrid_workflow.add_conditional_edges(
        START,
        route_prototype_workflow,
        {
            "react_main": "react_agent",
            "page_customization": "page_customization",
            "branding_extraction": "branding_extraction"
        }
    )

    # Sub-workflow returns
    hybrid_workflow.add_edge("page_customization", "react_agent")
    hybrid_workflow.add_edge("branding_extraction", "react_agent")
    hybrid_workflow.add_edge("react_agent", END)

    return hybrid_workflow.compile(checkpointer=MemorySaver())

def route_prototype_workflow(state: FincoderSwarmState) -> str:
    """Route to appropriate prototype workflow component"""

    if state.prototype.sub_workflow_active:
        return state.prototype.sub_workflow_type

    # Check if branding data needed
    if not state.branding_data.is_complete and state.bank_data and state.bank_data.webs
        return "branding_extraction"

    return "react_main"
```

Enhanced Prototype Tools:

```python
def get_available_pages_tool(
    tool_call_id: Annotated[str, InjectedToolCallId] = None,
) -> Command:
    """Retrieve available pages for customization"""
    pages_data = fetch_available_pages()

    pages = [
        PageModel(
            page_id=page["id"],
            name=page["name"],
            microapp=page["microapp"],
            version=page["version"],
            html_raw=page["html_raw"],
            html_elements=[HTMLElement(**elem) for elem in page.get("elements", [])],
            updated_at=page["updated_at"],
            introduction=page.get("introduction", ""),
            predefined_questions=[Question(**q) for q in page.get("questions", [])]
        )
        for page in pages_data
    ]

    return Command(
        update={
            "prototype": {
                **state.get("prototype", {}),
                "available_pages": pages,
                "awaiting_user_input": True,
                "input_type": "page_selection"
            },
            "workflow_status": WorkflowStatus(
                current_phase="page_selection",
                progress_percentage=30.0,
                next_recommended_action="select_page_to_customize"
            ),
            "messages": [
                ToolMessage(
                    content=f"Retrieved {len(pages)} available pages for customization"
                    tool_call_id=tool_call_id,
                )
            ],
        }
    )
```

```python
def html_change_planner_tool(
    user_request: str,
    selected_page: PageModel,
    tool_call_id: Annotated[str, InjectedToolCallId] = None,
) -> Command:
    """Plan HTML changes based on user request"""

    # Analyze user request and create step plan
    steps = plan_html_changes(user_request, selected_page)

    customization_steps = [
        CustomizationStep(
            step_id=f"step_{i+1}",
            description=step["description"],
            tools_required=step["tools"],
            expected_outcome=step["outcome"]
        )
        for i, step in enumerate(steps)
    ]

    return Command(
        update={
            "prototype": {
                **state.get("prototype", {}),
                "steps": customization_steps,
                "current_step_id": customization_steps[0].step_id if customization_step
            },
            "messages": [
                ToolMessage(
                    content=f"Created {len(customization_steps)} customization steps",
                    tool_call_id=tool_call_id,
                )
            ],
        }
    )

def page_customization_handoff_tool(
    tool_call_id: Annotated[str, InjectedToolCallId] = None,
) -> Command:
    """Handoff to page customization sub-workflow"""

    return Command(
        update={
```

```
            "prototype": {
                **state.get("prototype", {}),
                "sub_workflow_active": True,
                "sub_workflow_type": "page_customization"
            },
            "processing_metadata": ProcessingMetadata(
                current_step="handoff_to_page_customization",
                execution_context="page_customization_workflow",
                timestamp=datetime.now().isoformat()
            ),
            "messages": [
                ToolMessage(
                    content="Transferring to page customization workflow",
                    tool_call_id=tool_call_id,
                )
            ],
        }
    )

def ui_highlight_elements_tool(
    element_ids: List[str],
    tool_call_id: Annotated[str, InjectedToolCallId] = None,
) -> Command:
    """Highlight elements in UI for user interaction"""

    # Trigger UI highlighting
    highlight_result = trigger_ui_highlights(element_ids)

    return Command(
        update={
            "prototype": {
                **state.get("prototype", {}),
                "highlighted_elements": element_ids,
                "awaiting_user_input": True,
                "input_type": "element_selection"
            },
            "messages": [
                ToolMessage(
                    content=f"Highlighted {len(element_ids)} elements in UI",
                    tool_call_id=tool_call_id,
                )
            ],
```

```
    }
)
```

3.3. Page Customization Sub-Workflow

```
def build_page_customization_workflow():
    """Build page customization sub-workflow"""

    workflow = StateGraph(
        state_schema=PageCustomizationState,
        input_schema=FincoderSwarmState,
        output_schema=FincoderSwarmState,
    )

    # Entry: Convert to specialized state
    workflow.add_node("entry", page_customization_entry_node)

    # Processing nodes
    workflow.add_node("execute_steps", execute_customization_steps)
    workflow.add_node("apply_changes", apply_element_changes)
    workflow.add_node("get_confirmation", get_changes_confirmation)
    workflow.add_node("finalize_changes", finalize_customization)

    # Cleanup and exit
    workflow.add_node("cleanup", page_customization_cleanup)
    workflow.add_node("exit", page_customization_exit_node)

    # Flow definition
    workflow.add_edge(START, "entry")
    workflow.add_edge("entry", "execute_steps")

    workflow.add_conditional_edges(
        "execute_steps",
        check_step_completion,
        {
            "continue": "apply_changes",
            "needs_confirmation": "get_confirmation",
            "completed": "finalize_changes"
        }
    )

    workflow.add_edge("apply_changes", "execute_steps")
    workflow.add_edge("get_confirmation", "apply_changes")
    workflow.add_edge("finalize_changes", "cleanup")
    workflow.add_edge("cleanup", "exit")
    workflow.add_edge("exit", END)
```

```
    return workflow.compile()

def page_customization_entry_node(state: FincoderSwarmState) -> PageCustomizationState:
    """Entry point: Convert to specialized state"""

    return PageCustomizationState(
        steps=state.prototype.steps,
        messages=state.prototype.messages,
        selected_page=state.prototype.selected_page,
        processing_started=True
    )

def page_customization_exit_node(state: PageCustomizationState) -> FincoderSwarmState:
    """Exit: Convert back with results"""

    response_message = AIMessage(
        content=state.final_thoughts,
        additional_kwargs={
            "applied_changes": [change.dict() for change in state.applied_changes],
            "customization_summary": f"Applied {len(state.applied_changes)} changes"
        }
    )

    return FincoderSwarmState(
        messages=state.messages + [response_message],
        active_agent="Prototype_Agent",

        # Update prototype state
        prototype=PrototypeStateSchema(
            selected_page=state.selected_page,
            applied_changes=state.applied_changes,
            sub_workflow_active=False,
            sub_workflow_type=None
        ),

        # Update action summary
        last_action_summary=f"Applied changes to page {state.selected_page.name}. {len(

        # Update workflow status
        workflow_status=WorkflowStatus(
            current_phase="customization",
            last_action="page_customization_completed",
```

```
            progress_percentage=80.0,
            next_recommended_action="continue_customization_or_finalize"
        )
    )

def page_customization_cleanup(state: PageCustomizationState) -> PageCustomizationState
    """Mandatory cleanup before exit"""
    return {
        **state,
        # Reset processing flags
        "processing_started": False,
        "current_step_index": 0,
        "needs_cleanup": False,

        # Clear temporary data
        "current_element": None,
        "change_plan": [],

        # Preserve results
        # "applied_changes": state["applied_changes"],
        # "final_thoughts": state["final_thoughts"]
    }
```

4. ENHANCED DYNAMIC PROMPT ENGINEERING

4.1. Main Agent Enhanced Prompt

File: `prompts/fincoder/main_agent.jinja2`

```
{% if user_data %}
## User Context & Session Management
- **User:** {{ user_data.name }} ({{ user_data.role }})
{% if bank_data %}
- **Bank:** {{ bank_data.bank_name }}
{% if bank_data.website %}
- **Website:** {{ bank_data.website }}
{% endif %}
{% endif %}
- **Email:** {{ user_data.email }}
- **Language:** {{ user_data.language }}

### Session Timing Context
{% if user_data.is_new_user %}
- **Status:** **New User** - First time using the system
- **Welcome Behavior:** Provide comprehensive introduction and system overview
{% else %}
- **Status:** **Returning User**
{% if user_data.interval_since_latest_update %}
- **Time Since Last Activity:** {{ user_data.interval_since_latest_update }} minutes ag
{% if user_data.interval_since_latest_update < 30 %}
- **Conversation Style:** Continue naturally - recent activity
{% elif user_data.interval_since_latest_update < 120 %}
- **Conversation Style:** Brief recap, then continue
{% else %}
- **Conversation Style:** Welcome back with context reminder
{% endif %}
{% endif %}
{% endif %}

{% if last_task_description %}
### Last Task Context
- **Previous Task:** {{ last_task_description }}
- **Continuation:** Reference this context for seamless experience
{% endif %}

{% if last_action_summary %}
### Last Action Summary
- **Recent Activity:** {{ last_action_summary }}
{% endif %}
{% endif %}

{% if system_capabilities %}
```

```
## Available System Capabilities
{% for capability in system_capabilities %}
- **{{ capability.name }}** → {{ capability.description }}
  - Entry Agent: {{ capability.entry_agent }}
  - Available: {{ "Yes" if capability.is_available else "No" }}
{% endfor %}
{% endif %}

{% if workflow_status %}
## Current Workflow Status
- **Phase:** {{ workflow_status.current_phase }}
- **Progress:** {{ workflow_status.progress_percentage }}%
{% if workflow_status.last_action %}
- **Last Action:** {{ workflow_status.last_action }}
{% endif %}
{% if workflow_status.next_recommended_action %}
- **Next Step:** {{ workflow_status.next_recommended_action }}
{% endif %}
{% endif %}

## Adaptive Conversation Rules

### New User Welcome Protocol
{% if user_data and user_data.is_new_user %}
**ACTIVE:** Provide comprehensive system introduction
1. **Welcome Message:** Personalized greeting with system overview
2. **Capabilities Presentation:** Show all available system capabilities
3. **Guidance:** Offer clear next steps and options
4. **System Context:** Explain Temenos Digital Banking customization focus
{% endif %}

### Returning User Continuation Protocol
{% if user_data and not user_data.is_new_user %}
**ACTIVE:** Provide contextual continuation
{% if user_data.interval_since_latest_update and user_data.interval_since_latest_update
**Recent Activity:** Continue conversation naturally without re-introduction
{% elif user_data.interval_since_latest_update and user_data.interval_since_latest_upda
**Medium Gap:** Brief "welcome back" with context recap
{% else %}
**Long Gap:** "Welcome back" with full context restoration and options
{% endif %}
{% endif %}
```

```
### Routing Decision Logic
1. **"Temenos Digital Customization"** → Transfer to `Prototype_Agent`
2. **"Project Scope Discovery"** → Transfer to `Scope_Discovery_Agent`
3. **"Continue Previous Task"** → Route based on `last_task_description`
4. **General Questions** → Handle directly with context awareness

## Response Format Guidelines
- Use conversational tone appropriate to session timing
- Provide numbered options for clear user choices
- Leverage user context for personalized experience
- Maintain professional but friendly demeanor
- Always offer clear next steps
```

4.2. Prototype Agent ReAct Component Prompt

File: `prompts/fincoder/prototype_react.jinja2`

```
{% if branding_data %}
## Branding Context
- **Primary Color:** {{ branding_data.primary_color or "Not set" }}
- **Secondary Color:** {{ branding_data.secondary_color or "Not set" }}
- **Bank Logo:** {{ branding_data.bank_logo or "Not set" }}
- **Source:** {{ branding_data.source }}
- **Complete:** {{ "Yes" if branding_data.is_complete else "No - Missing data" }}

{% if not branding_data.is_complete %}
### Branding Data Collection Required
{% if bank_data and bank_data.website %}
**Website Available:** {{ bank_data.website }}
- Use `branding_extraction_handoff_tool` to extract branding from website
{% else %}
**No Website:** Use `human_in_the_loop` to collect branding data manually
{% endif %}
{% endif %}
{% endif %}

{% if prototype.selected_page %}
## Active Page Context
- **Page:** {{ prototype.selected_page.name }} (ID: {{ prototype.selected_page.page_id
- **Microapp:** {{ prototype.selected_page.microapp }}
- **Version:** {{ prototype.selected_page.version }}
- **Elements:** {{ prototype.selected_page.html_elements|length }} customizable element
- **Predefined Questions:** {{ prototype.selected_page.predefined_questions|length }} q

{% if prototype.selected_page.predefined_questions %}
### Predefined Questions Status
{% for question in prototype.selected_page.predefined_questions %}
- **{{ question.question_text }}**
  - Type: {{ question.question_type }}
  - Required: {{ "Yes" if question.required else "No" }}
  - Status: {{ "Answered" if question.answered else "Pending" }}
{% if question.answered %}
  - Answer: {{ question.answer }}
{% endif %}
{% endfor %}

{% if prototype.selected_page.get_unanswered_questions() %}
**Action Required:** Process unanswered questions using `human_in_the_loop`
{% endif %}
{% endif %}
```

```
{% elif prototype.available_pages %}
## Available Pages
{{ prototype.available_pages|length }} pages available for customization:
{% for page in prototype.available_pages[:5] %}
- **{{ page.name }}** ({{ page.microapp }}) - {{ page.html_elements|length }} elements
{% endfor %}
{% if prototype.available_pages|length > 5 %}... and {{ prototype.available_pages|lengt

**Action Required:** User needs to select a page for customization
{% else %}
## Page Selection Required
Use `get_available_pages_tool` to retrieve available pages for customization
{% endif %}

{% if prototype.steps %}
## Customization Plan
Current plan has {{ prototype.steps|length }} steps:
{% for step in prototype.steps %}
{{ loop.index }}. **{{ step.description }}**
   - Tools: {{ step.tools_required|join(", ") }}
   - Expected: {{ step.expected_outcome }}
   - Status: {{ "Completed" if step.completed else "Pending" }}
{% endfor %}

{% if not prototype.sub_workflow_active %}
**Ready for Execution:** Use `page_customization_handoff_tool` to execute plan
{% endif %}
{% endif %}

{% if prototype.highlighted_elements %}
## UI Interaction State
- **Highlighted Elements:** {{ prototype.highlighted_elements|length }} elements highli
- **Awaiting Input:** {{ "Yes" if prototype.awaiting_user_input else "No" }}
- **Input Type:** {{ prototype.input_type or "None" }}
{% endif %}

{% if prototype.sub_workflow_active %}
## Sub-Workflow Active
- **Type:** {{ prototype.sub_workflow_type }}
- **Status:** Sub-workflow is currently processing
- **Wait:** Allow sub-workflow to complete before taking action
{% endif %}
```

Tool Usage Strategy

Page Management Flow
1. **Check Branding:** Ensure branding data is complete first
2. **Load Pages:** Use `get_available_pages_tool()` if needed
3. **User Selection:** Use `human_in_the_loop` for page selection
4. **Process Questions:** Handle predefined questions if any

Element Interaction Flow
1. **Get Elements:** Use `get_element_by_id_tool` or `find_elements_tool`
2. **Visual Feedback:** Use `get_screenshot_tool` for visual context
3. **User Guidance:** Use `ui_highlight_elements_tool` for interaction
4. **Plan Creation:** Use `html_change_planner_tool` for structured plan

Sub-Workflow Orchestration
1. **Plan Ready:** Ensure customization steps are defined
2. **Transfer Control:** Use `page_customization_handoff_tool`
3. **Wait for Results:** Sub-workflow will return with applied changes
4. **Continue:** Handle user feedback or additional requests

Interaction Patterns

When Page Not Selected
- Use `get_available_pages_tool` if not already loaded
- Present options clearly via `human_in_the_loop`
- Provide page descriptions and capabilities

When Questions Exist
- Process all predefined questions sequentially
- Use clear, specific questions via `human_in_the_loop`
- Store answers appropriately in page state

When User Requests Changes
- Use `html_change_planner_tool` to create structured plan
- Show plan to user for confirmation
- Execute via `page_customization_handoff_tool`

Visual Element Identification
- Use `find_elements_tool` for discovery
- Use `ui_highlight_elements_tool` for visual guidance
- Use `get_screenshot_tool` for confirmation
- Get user confirmation before proceeding

```
## Error Handling
- Always validate page selection before proceeding
- Ensure branding data completeness
- Handle missing elements gracefully
- Provide clear feedback for user actions
```

Com esses refinamentos baseados nos cenários reais, o documento agora reflete:

1. **Estado híbrido** - ReAct + Custom Workflow combination
2. **Gestão de sessão adaptativa** - Comportamento baseado em timing
3. **Sub-workflows especializados** - Page customization e branding extraction
4. **Estados estruturados** - PageModel, HTMLElement, CustomizationStep
5. **Ferramentas realistas** - UI highlighting, element manipulation, change planning
6. **Prompts adaptativos** - Baseados em contexto de sessão e estado atual

Está pronto para os próximos cenários para continuarmos refinando!

5. ARQUITETURA DE WORKFLOW: ESTADOS, FLUXOS E CONEXÕES

5.1. Rastreabilidade de Workflow: PageChanges ↔ CustomizationStep

Especificação de Rastreabilidade:

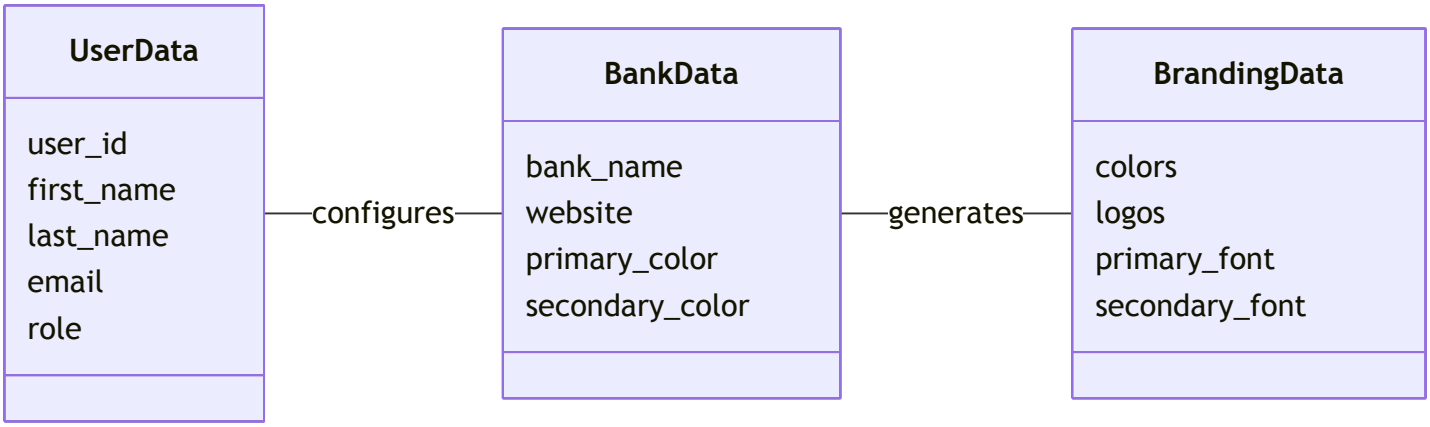
A arquitetura implementa **rastreabilidade completa** entre mudanças aplicadas (`PageChanges`) e os steps de workflow (`CustomizationStep`) que as originaram. Esta especificação garante auditoria completa e contexto preservado em todas as operações de customização.

```python
# ESPECIFICAÇÃO: PageChanges com rastreabilidade completa
class PageChanges(BaseModel):
    change_id: str
    element_id: str
    change_data: HtmlChangeUnion
    timestamp: str
    applied: bool = False

        # RASTREABILIDADE DE WORKFLOW IMPLEMENTADA
     step_id: str = Field(description="ID do CustomizationStep que gerou esta mudança")
     step_description: str = Field(description="Descrição do step para contexto")
     execution_order: int = Field(description="Ordem de execução dentro do step")

     # CONTEXTO DE AUDITORIA E CONTROLE DE ESTADO
     application_status: Literal["pending", "applied", "failed", "rolled_back"] = "pend
     error_message: Optional[str] = Field(description="Erro se aplicação falhou")
     rollback_data: Optional[str] = Field(description="Dados para rollback se necessári
```

5.2. Arquitetura Completa: Storytelling em 5 Camadas

Camada 1: Dados Fundamentais (Foundation Layer)

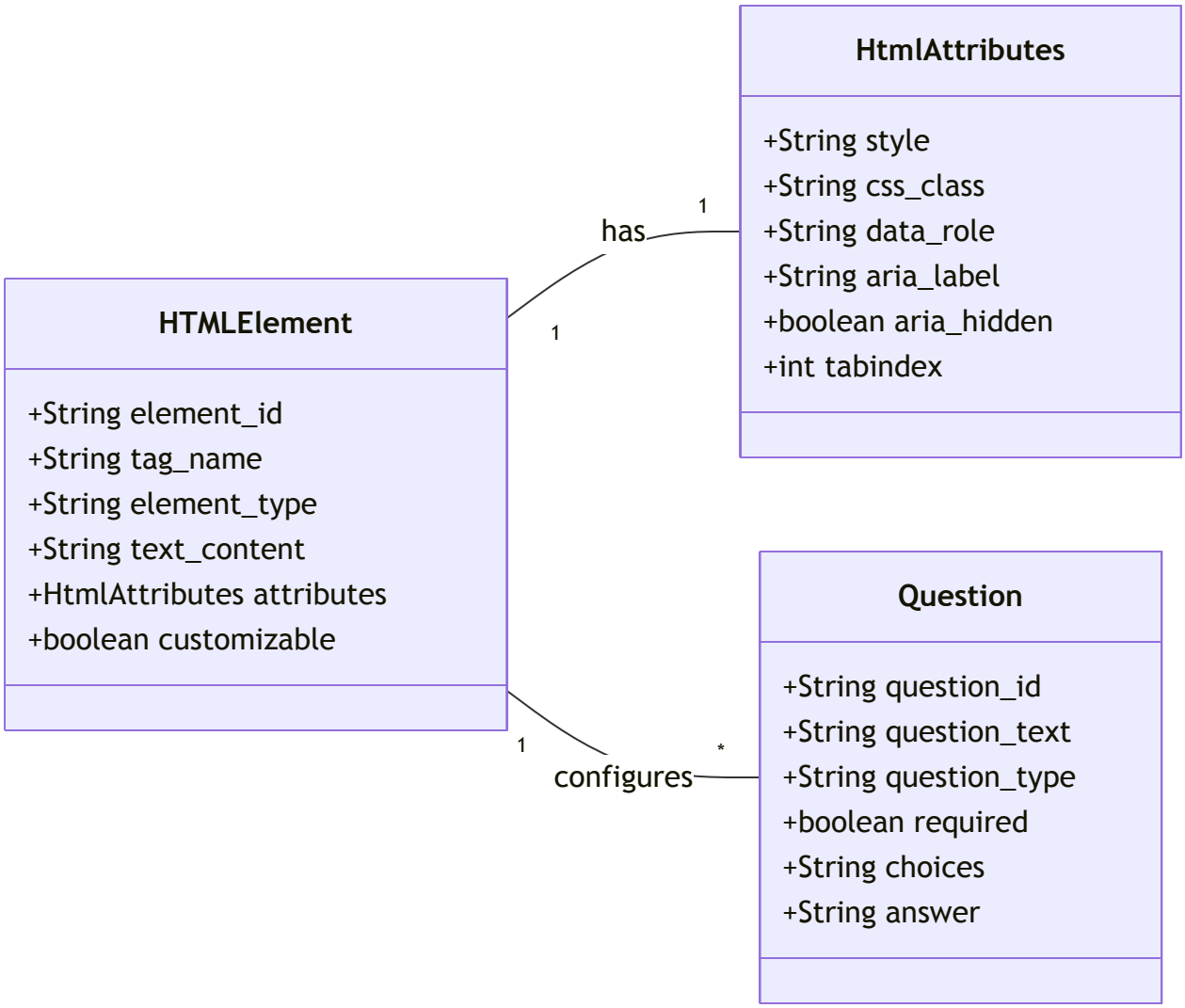
Propósito: Representar informações básicas do domínio bancário e usuário



História: Um usuário (`UserData`) configura dados do banco (`BankData`) que são usados para extrair informações de branding (`BrandingData`) do website institucional.

Camada 2: Estruturas HTML (Content Layer)

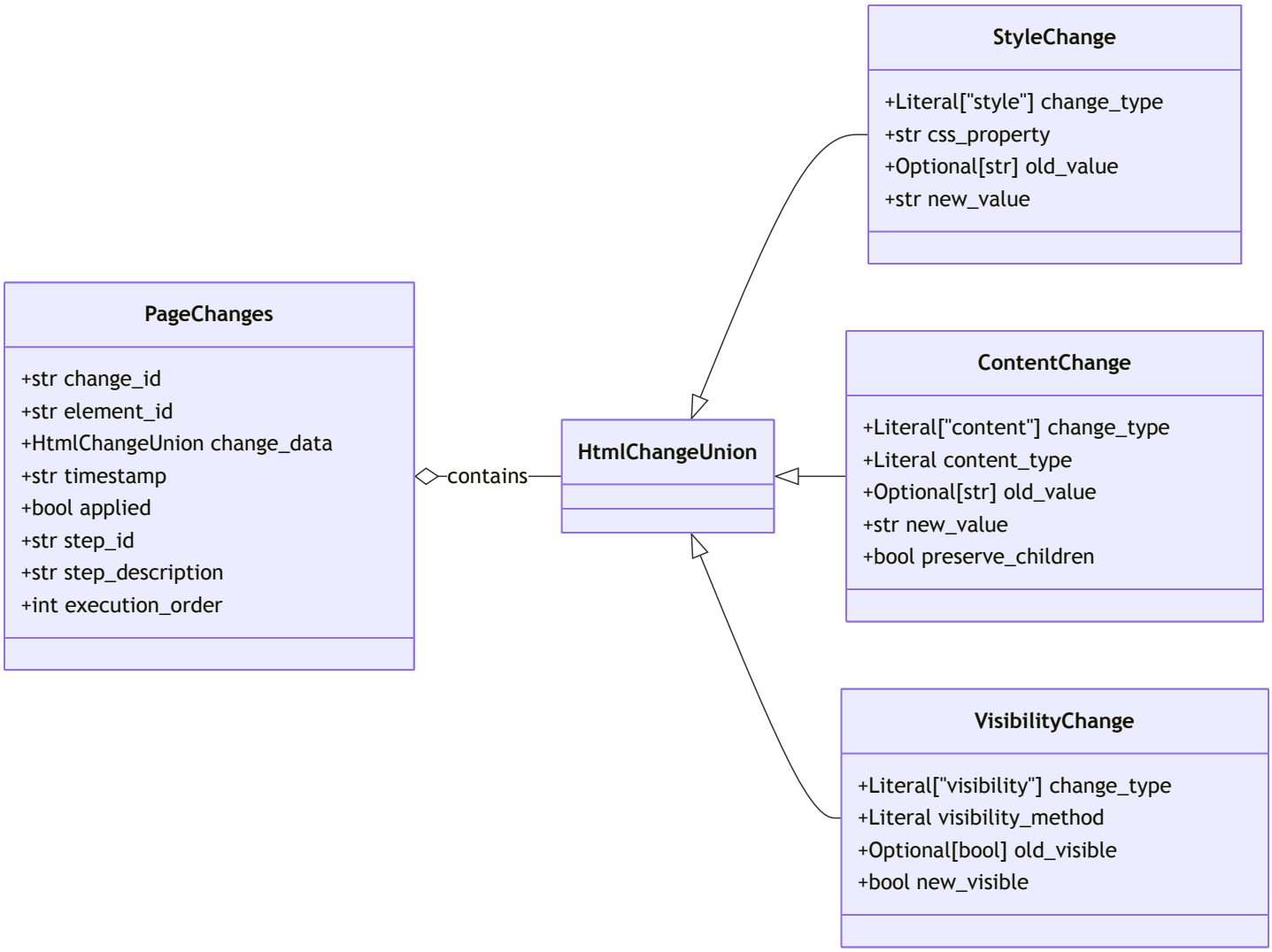
Propósito: Representar elementos de página que podem ser modificados



História: Elementos HTML (`HTMLElement`) possuem atributos estruturados (`HtmlAttributes`) e podem ser configurados através de perguntas (`Question`) feitas ao usuário.

Camada 3: Mudanças e Transformações (Change Layer)

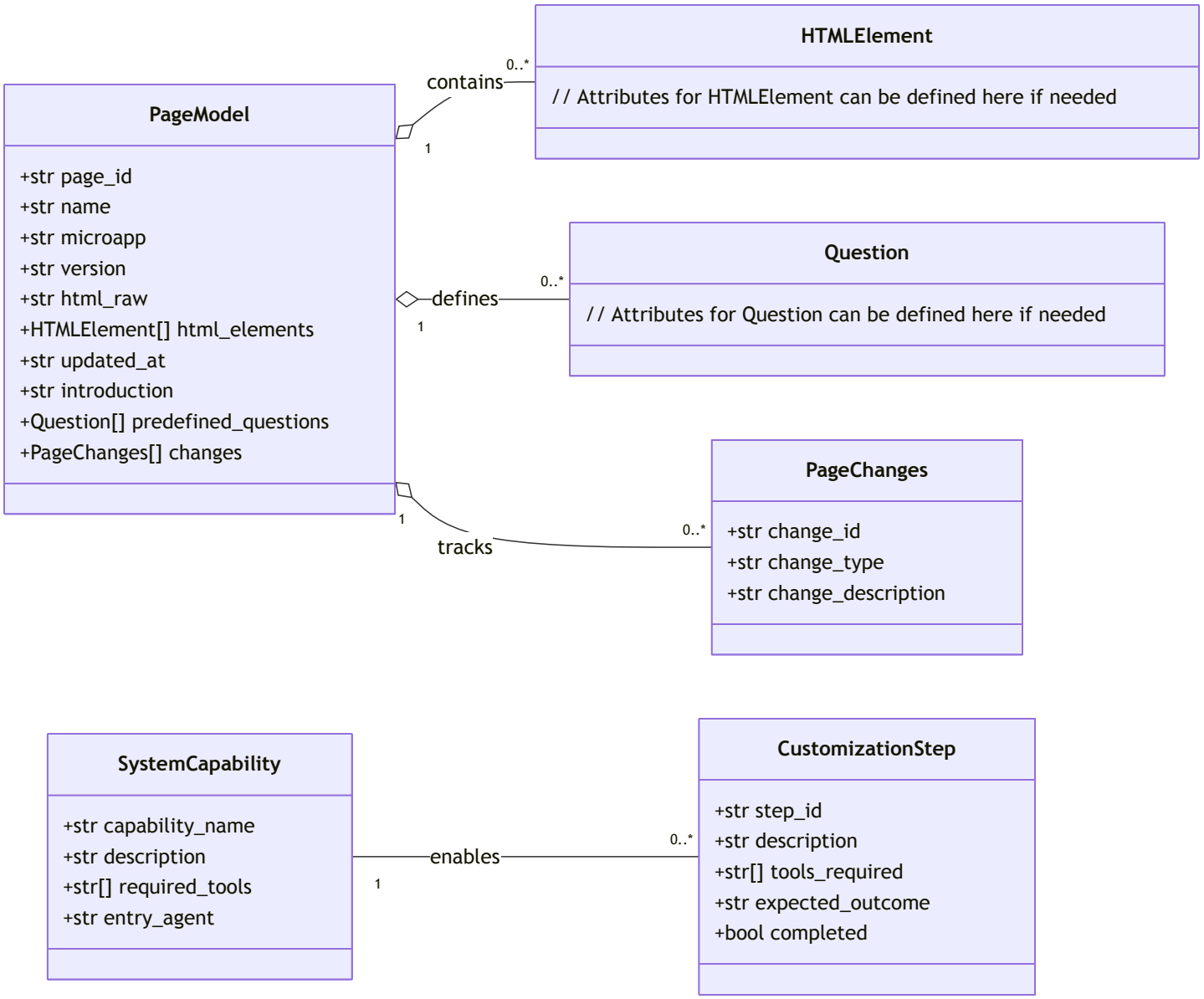
Propósito: Representar transformações aplicadas aos elementos com Union Types discriminados



História: Mudanças específicas (`StyleChange` , `ContentChange` , `VisibilityChange`) são encapsuladas em `PageChanges` que conecta as transformações com o contexto do workflow.

Camada 4: Workflow e Steps (Process Layer)

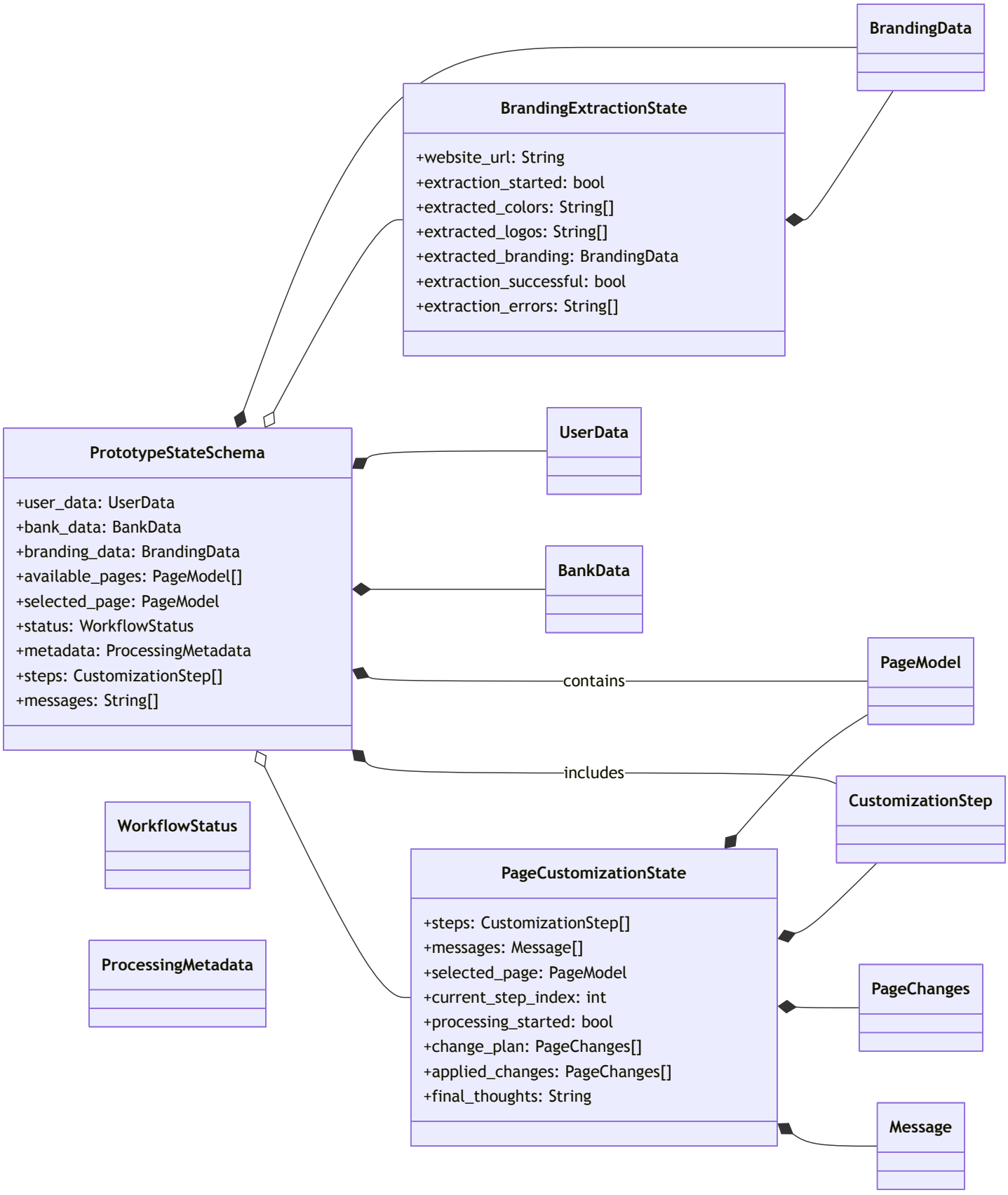
Propósito: Orquestrar a execução de mudanças através de steps estruturados



História: Steps de customização (`CustomizationStep`) geram mudanças (`PageChanges`) que são aplicadas a páginas (`PageModel`) usando capacidades do sistema (`SystemCapability`).

Camada 5: Estados de Workflow (State Layer)

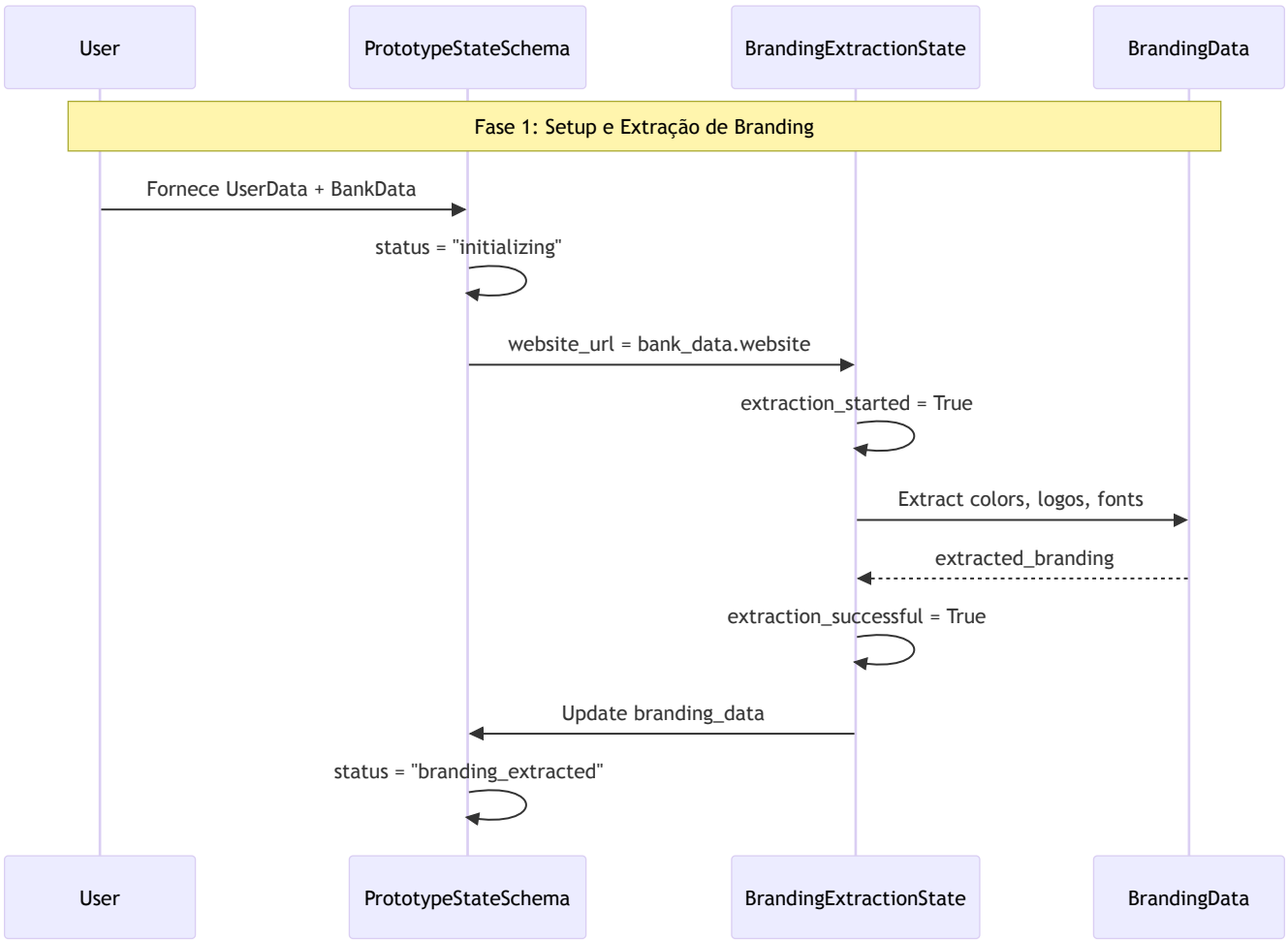
Propósito: Gerenciar estado e progressão através dos workflows



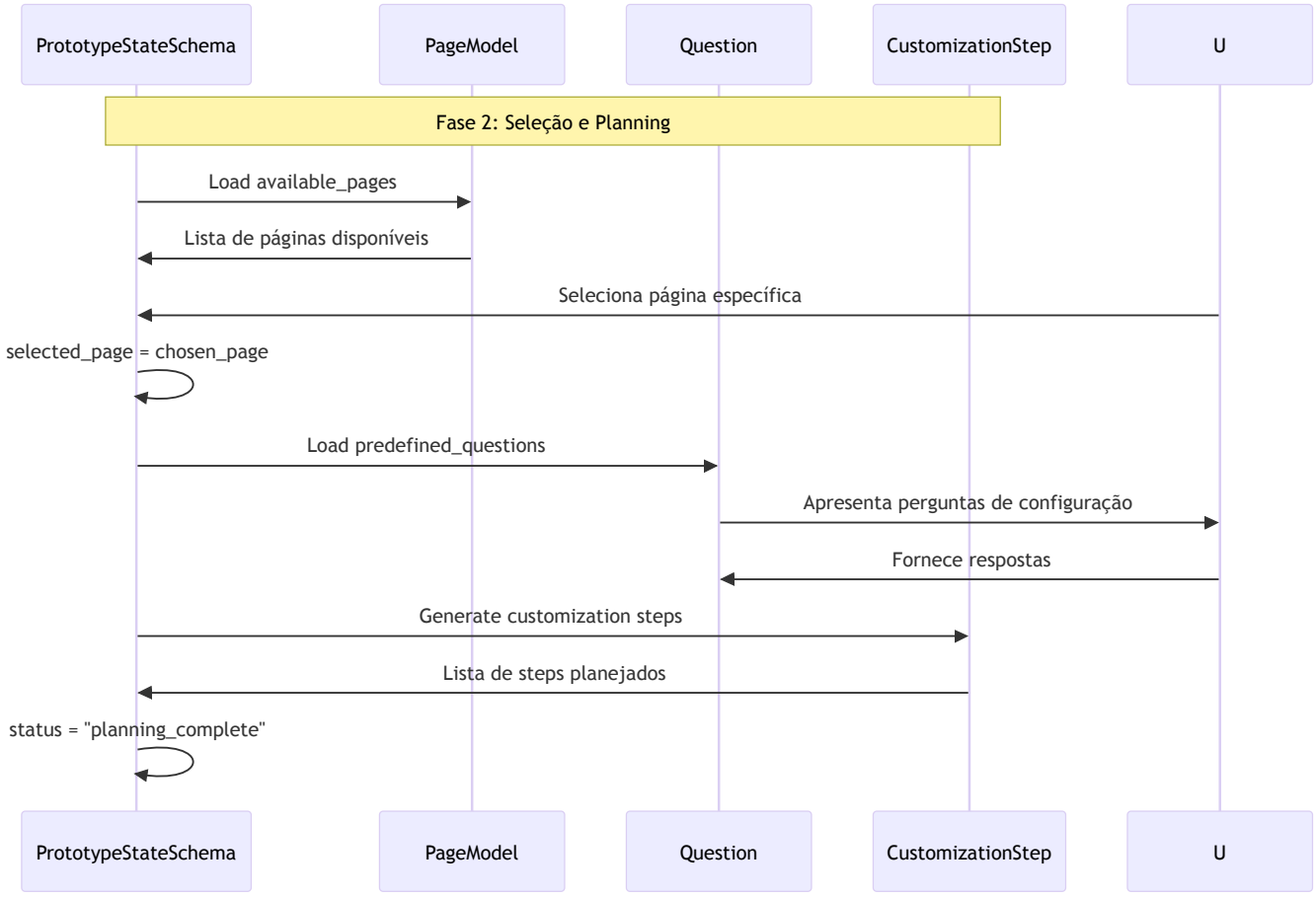
História: Estados principais (`PrototypeStateSchema`) orquestram sub-workflows (`PageCustomizationState` , `BrandingExtractionState`) que executam operações específicas e mantêm tracking detalhado de progresso.

5.3. Fluxos de Execução Completos

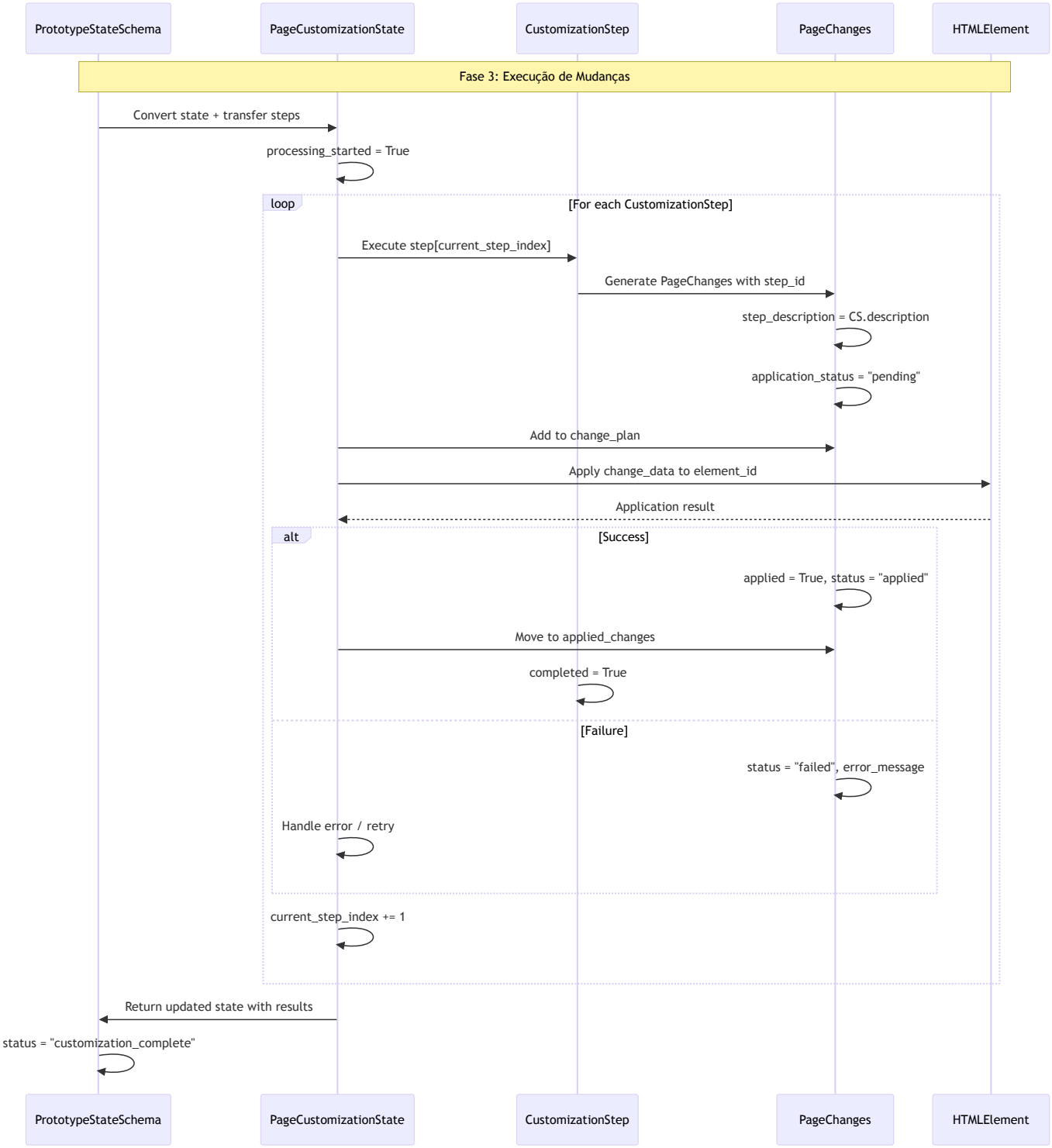
Sequência 1: Inicialização e Setup



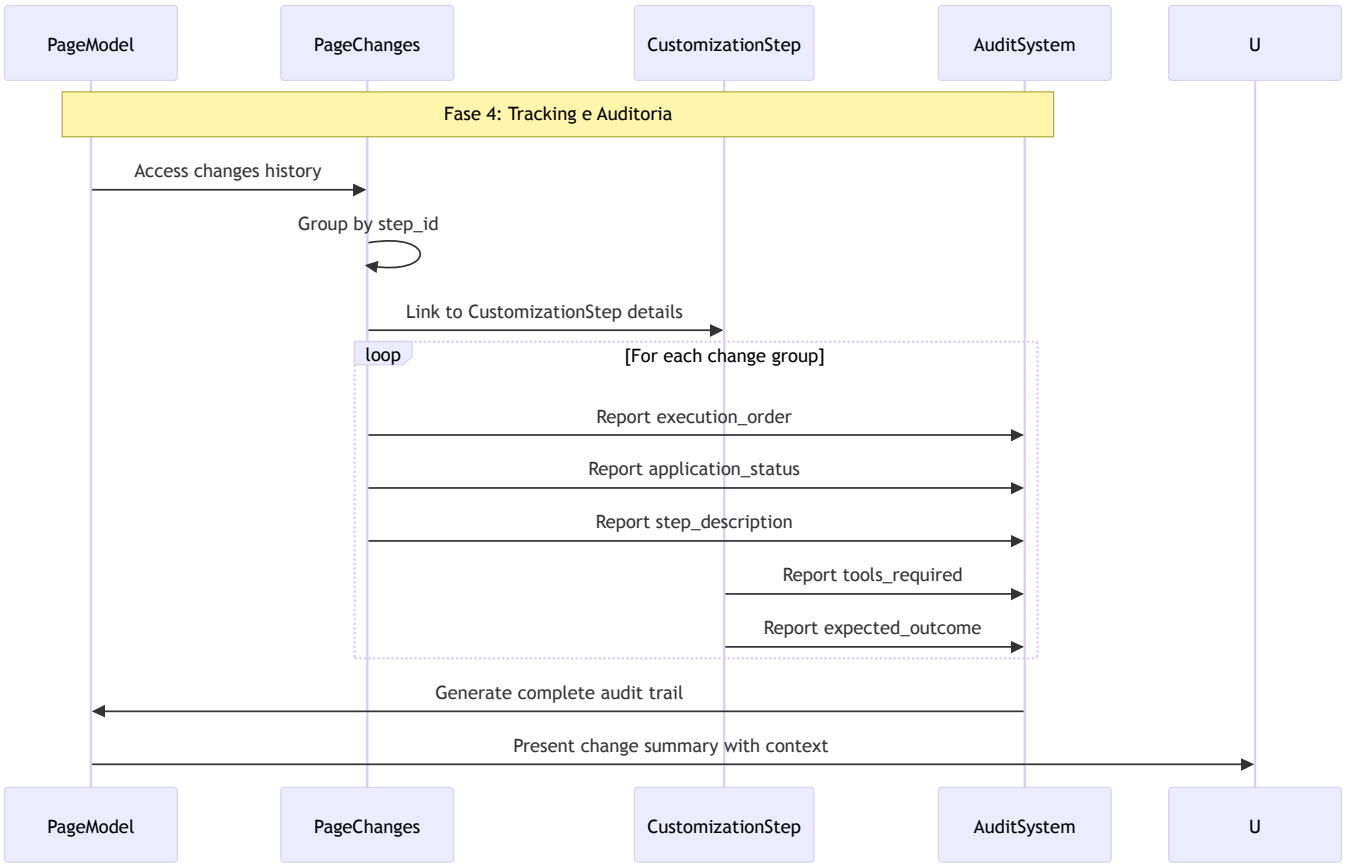
Sequência 2: Seleção de Página e Planning



Sequência 3: Execução de Customização



Sequência 4: Tracking e Auditoria



5.4. Storytelling Completo: A Jornada do Workflow Multi-Agente

Ato I: O Usuário e Sua Necessidade

A história começa quando um usuário (`UserData`) acessa o sistema com a necessidade de customizar uma página bancária no Temenos Digital. Ele fornece dados básicos do banco (`BankData`) incluindo o website institucional. O sistema então inicia um sub-workflow de extração (`BrandingExtractionState`) que analisa o website e extrai cores, logos e fontes, criando um perfil de branding (`BrandingData`) completo e estruturado.

Ato II: Descoberta e Planejamento

Com o branding extraído, o sistema apresenta páginas disponíveis (`PageModel`) que contêm elementos customizáveis (`HTMLElement`) com atributos estruturados (`HtmlAttributes`). O usuário responde perguntas predefinidas (`Question`) que orientam o processo de customização. Baseado nessas respostas e no branding extraído, o sistema gera uma série de steps de customização (`CustomizationStep`) que descrevem exatamente o que será modificado, com que ferramentas, e qual resultado esperado.

Ato III: Execução e Transformação

O workflow entra na fase de execução (`PageCustomizationState`) onde cada step é processado sequencialmente. Para cada step, o sistema gera mudanças específicas (`PageChanges`) que incluem transformações detalhadas através de union types discriminados (`StyleChange` , `ContentChange` , `VisibilityChange`). **Crucialmente**, cada mudança está agora conectada ao step que a gerou através de `step_id` , `step_description` e `execution_order` , criando rastreabilidade completa.

Ato IV: Aplicação e Validação

As mudanças são aplicadas aos elementos HTML com tracking detalhado de status (`application_status`). Se uma aplicação falha, o erro é capturado (`error_message`) e dados de rollback são preservados (`rollback_data`). O sistema mantém listas separadas de mudanças planejadas (`change_plan`) e aplicadas (`applied_changes`), permitindo comparação e auditoria completa.

Ato V: Auditoria e Conclusão

Ao final, o sistema possui uma trilha completa de auditoria onde cada mudança pode ser rastreada até o step específico que a originou, com contexto completo sobre tools utilizadas, outcomes esperados, e status de execução. Isso permite debugging eficiente, rollbacks seletivos, e relatórios detalhados para o usuário final.

5.5. Características Arquiteturais

1. Rastreabilidade Completa

- Cada `PageChanges` conectado ao `CustomizationStep` que o gerou
- Ordem de execução preservada com `execution_order`
- Contexto disponível através de `step_description`
- História completa de mudanças com contexto de step

2. Workflow Management Robusto

- Estados hierárquicos: `PrototypeStateSchema` → `PageCustomizationState` → `PageChanges`
- Tracking de progresso por step e por mudança individual
- Error handling e rollback capabilities com `rollback_data`
- Status detalhado de cada aplicação (`pending` , `applied` , `failed` , `rolled_back`)

3. Auditoria e Debugging Avançados

- Capability de conectar falhas com steps específicos
- Mensagens de erro estruturadas (`error_message`)
- Contexto de execução preservado (`step_description`)
- Timeline completa de modificações com timestamps

4. Type Safety e Validation

- Union types discriminados para mudanças específicas
- Estruturas Pydantic para validação automática
- Zero dicionários genéricos em toda a arquitetura
- Contratos estruturados para todas as interfaces

5. Escalabilidade e Manutenibilidade

- Modelos modulares que podem evoluir independentemente
- Conexões bem definidas entre camadas
- Separation of concerns clara entre dados, workflow e execução
- Base sólida para adicionar novos tipos de customização

5.6. Capacidades da Arquitetura

A arquitetura implementa um **sistema de workflow inteligente** com rastreabilidade total em todas as operações. Esta especificação estabelece uma base enterprise-grade que oferece:

- **Debugging**: Conectar falhas específicas aos steps que as geraram
- **User Experience**: Fornecer feedback contextual sobre progresso de customização
- **Compliance**: Manter trilha de auditoria completa para todas as modificações
- **Maintenance**: Facilitar evolução e debugging do sistema multi-agente
- **Rollback**: Permitir desfazer mudanças em ordem correta com contexto preservado
- **Extensibilidade**: Base para adicionar novos tipos de workflows e agentes especializados

Esta arquitetura especifica uma **solução enterprise-grade** com rastreabilidade total, estabelecendo a base para **workflows multi-agente robustos** e **totalmente auditáveis** no contexto de banking applications.