Seasar Conference 2006 Autumn

JBoss jBPMと Seasar2の連携

水野浩典・佐藤修一
オープンソース・コンピテンシ・センター
日本ヒューレットパッカード株式会社

2006年11月12日

目的

- JBossから提供されているワークフロー管理システムである jBPM（Java Business Process Management）の概要

- jBPMとSeasarの連携

内容

- JBoss jBPMとは？
- JBoss jBPM システム概要
- プロセスの動作概要
- jBPMアプリケーション
- jBPM webアプリケーション

- Seasar2との連携

JBoss jBPMとは？

JBoss jBPM製品概要

- 全てJavaで作られたワークフロー管理システム
- オープンソース、ライセンスはLGPL
- ライセンス費用が0円
- シンプルなコアエンジン
- 他Javaライブラリへの依存度が非常に低い
  - J2EEアプリケーションサーバは無くても動作可能
- JDBC対応の任意のデータベースサーバに対応
- ワークフロー(ビジネスプロセス)を図で表現可能
- ビジネスプロセスをXML(JPDL)で定義
  - JPDL : jBPM プロセス定義言語
- BPELをサポート
- 既存システムとの連携
- 認証サーバ(LDAPサーバ等)との連携機能

- Eclipseを用いた開発
  - GPDプラグインによるビジネスプロセスのGUI開発、定義
    - GPD : グラフィカル・プロセス・デザイナー
    - プロセスのGUI表現とJPDL(XML) 間の相互変換が瞬時に可能
- Ant, JUnit等の汎用ツールで開発・デプロイ・テスト可能
- 標準化への取り組み
  - JBoss jBPM の開発リーダーやJBoss開発チームがJSRに参画
    - JSR 207: Process Definition for Java
    - JSR 208: Java Business Integration (JBI)
    - JSR 299: Web Beans

事例（海外の事例）

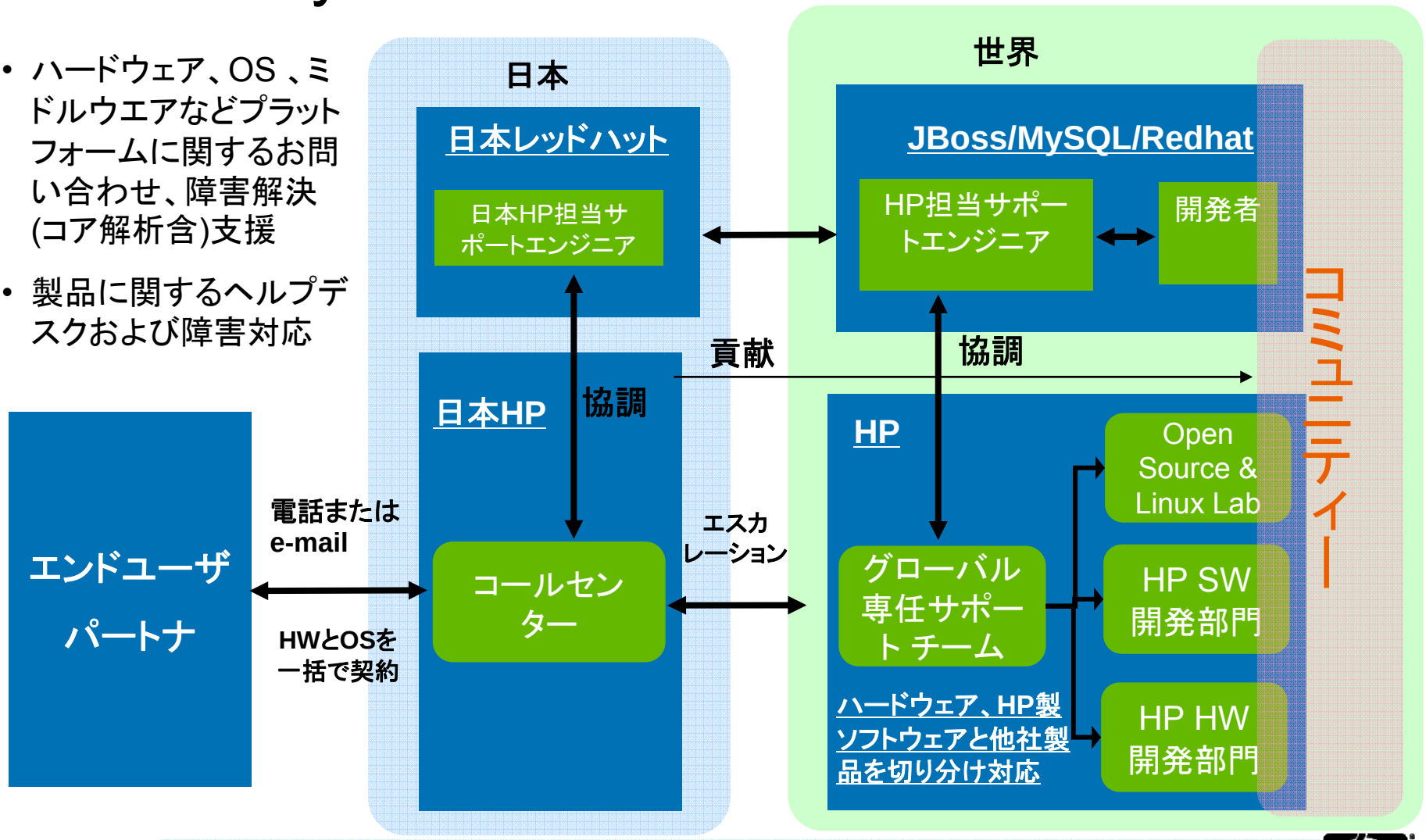
- **事例(1)**
  - 会社名：大手保険会社
  - 業界：金融
  - 採用目的：クレーム申請、新規保険証書の発行処理
  - 性能：100,000 プロセス・ステップ/分
- **事例(2)**
  - 会社名：大手航空宇宙会社
  - 業界：運輸
  - 採用目的：調達承認処理
    - 部品納入ベンダが登録する
    - 大手航空宇宙会社による仕様確認やベンダへの質問
- **事例(3)**
  - 会社名：大手金融サービス会社
  - 業界：金融
  - 採用目的：証券取引所における不正処理調査
- **事例(4)**
  - 会社名：大手政府機関
  - 業界：政府
  - 採用目的：法的処置案件管理システムのプロトタイプ
- **事例(5)**
  - 会社名：Raible Designs社
  - 採用目的：JBoss jBPMを使ったESB(Enterprise Service Bus)
    - 参照URL：http://raibledesigns.com/page/rd?anchor=djug_building_an_open_source
- **事例(6)**
  - 会社名：ルクセンブルク陸軍
  - 業界：政府
  - 採用目的：JBoss jBPMを使ったERP
- **事例(7)**
  - 会社名：司法サービス会社
  - 業界：法曹界
  - 採用目的：Windowsのクライアント/サーバによる貧弱なワークフローシステムをJBoss jBPMを使った Webアプリケーションに置き換え
- **事例(8)**
  - 会社名：Computation社
  - 業界：不明
  - 採用目的：ERPシステムのワークフロー処理
- **事例(9)**
  - 会社名：Met@Logo (ドイツ技術協力公社(GTZ)による eGovernmentプロジェクト)
    - www.metalogo.org
  - 業界：政府
  - 採用目的：南米各国の地方自治体向け電子政府ソリューションにおけるビジネスプロセス処理の基盤エンジンとして
- **事例(10)**
  - 会社名：Texas Institute of Genomic Medicine
  - 業界：バイオ
  - 採用目的：ビジネス処理のモデル化として採用

HPにおけるJBoss

- HPワールドワイドで製品保守を提供
  - JBoss Application Server
  - JBoss jBPM
  - JEMS(JBoss Enterprise Middleware Suite)
- 日本HPでの対応
  - JBoss Application Server
    - 2005年7月から標準時間の製品保守を提供
  - JBoss jBPM, JEMS
    - 現在準備中
  - JBoss jBPM開発支援コンサルティングを提供

オープンソースサポート体制（JBoss/MySQL/Redhat）

- ハードウェア、OS 、ミドルウエアなどプラットフォームに関するお問い合わせ、障害解決(コア解析含)支援
- 製品に関するヘルプデスクおよび障害対応



HP と 各社の強固なパートナーシップによる問題解決体制

JBoss開発者支援コンサルティング

全工程に対する一貫した開発者支援サービス

経験不足

- 新しいソフトウェアに対する経験不足

＋

情報不足

- ドキュメントや、システム構築のノウハウ、トラブルシューティングなどに関する情報不足

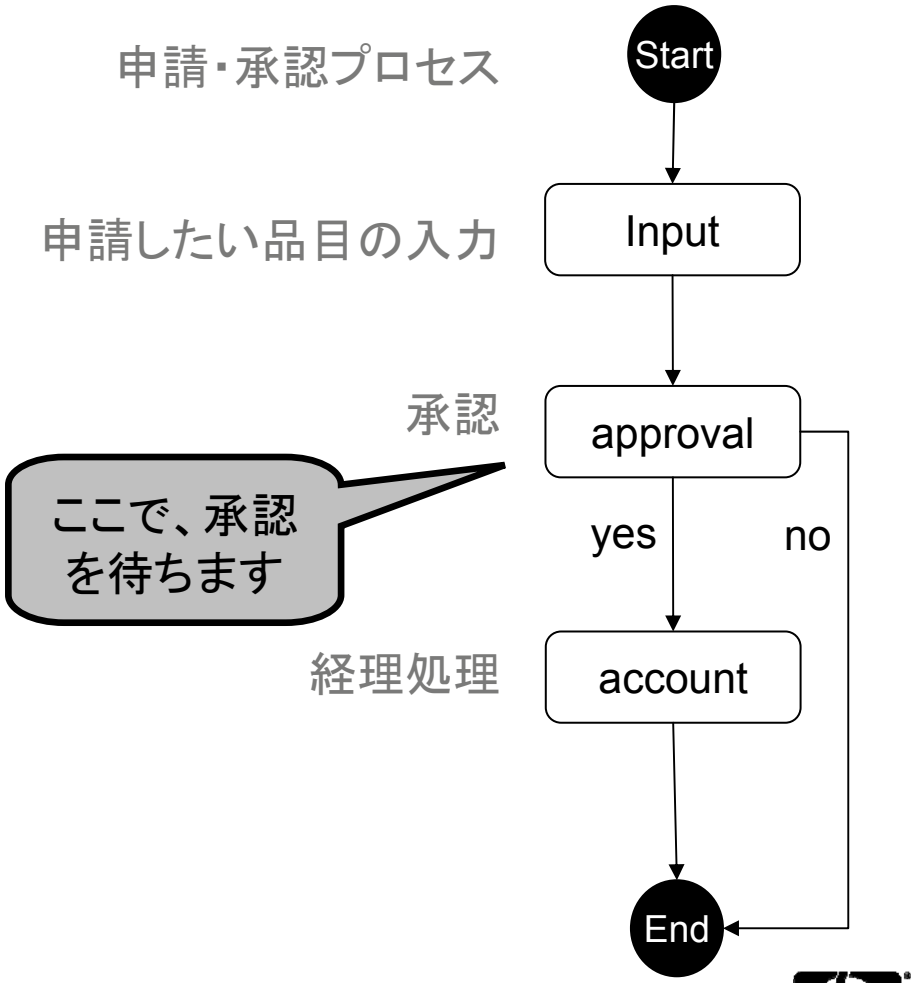
サービス内容

- 設計支援
- 仕様決定支援
- 障害解析支援
- サンプルコード提供
- 運用手順支援

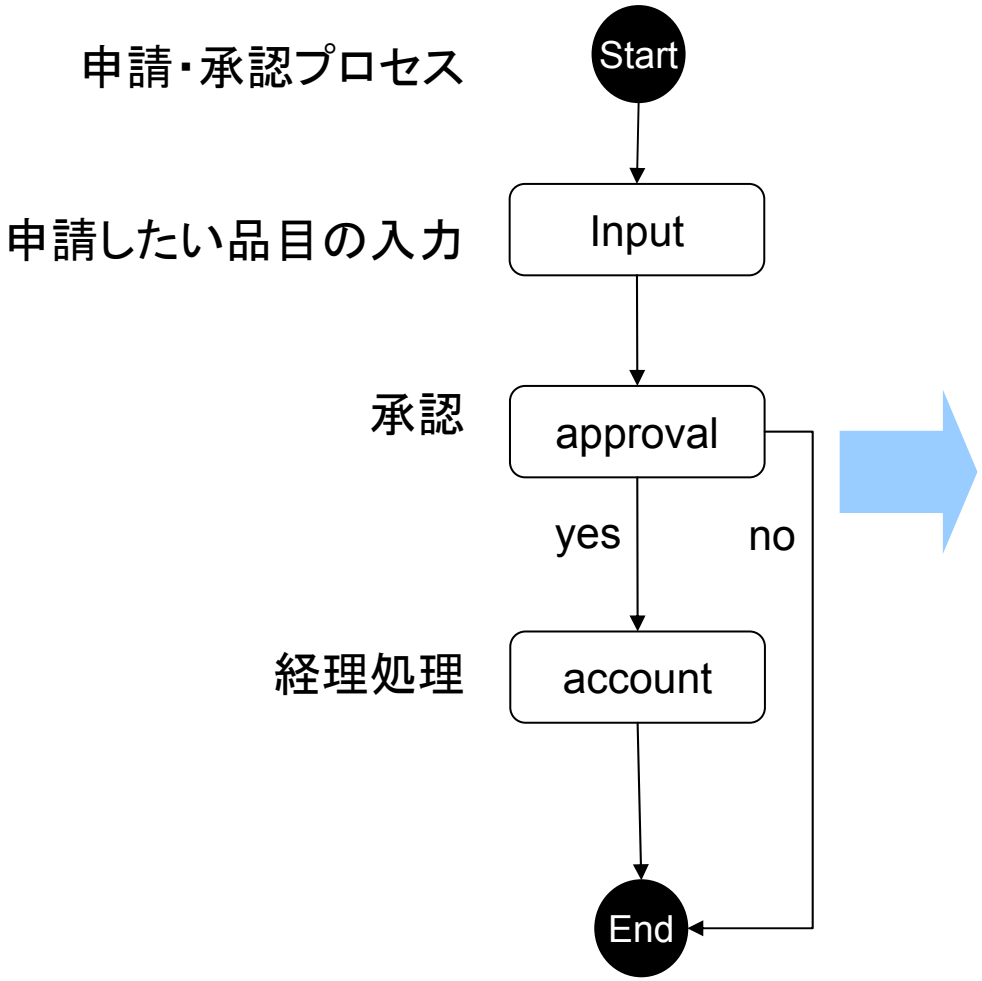
JBoss jBPMシステム概要

プロセス

- ビジネスなどの処理フローを表現したもの
  - 通常のプログラムのフローチャートとは異なります。
- 承認などの外部からのイベントを待つことが可能
- プロセスはJPDLを用いて記述・表現する



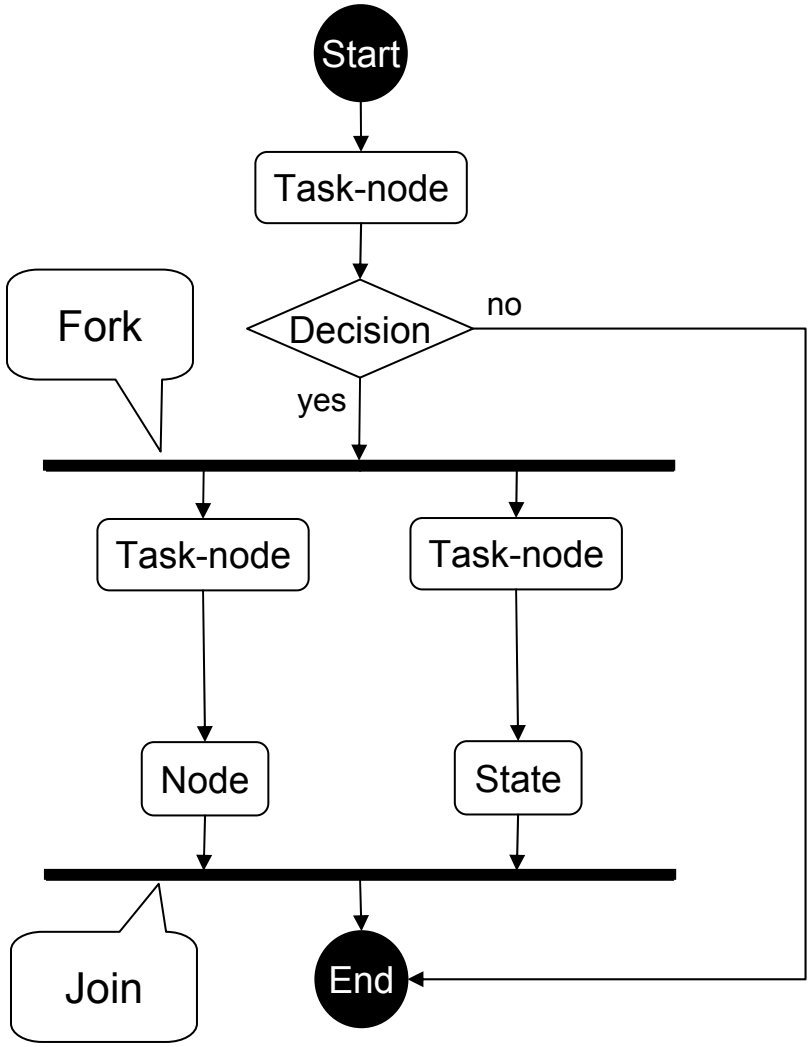
プロセス定義ファイル(JPDL)



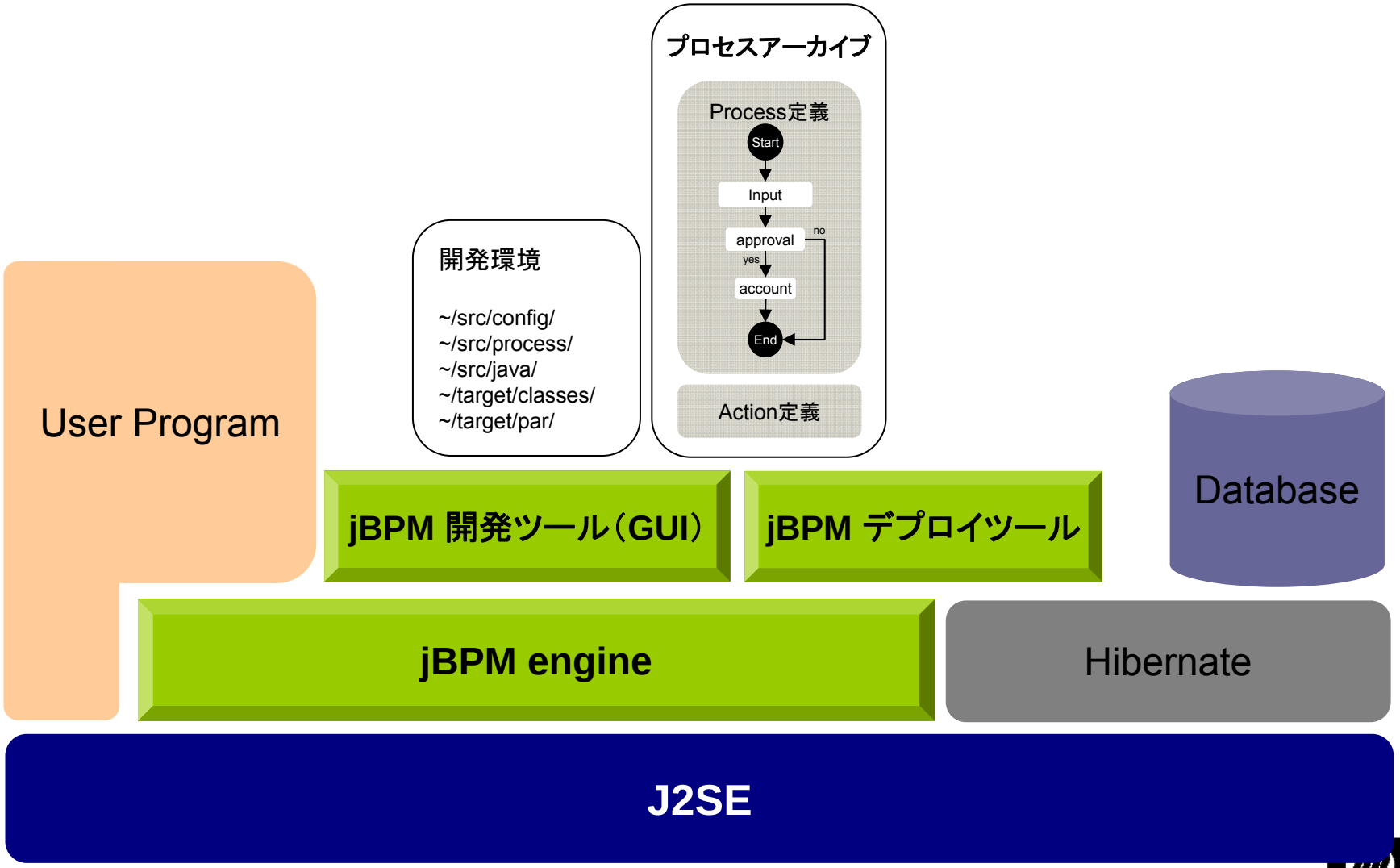

```
<process-definition name="pay raise process">

  <start-state name="Start">
    <transition to="input" />
  </start-state>

  <task-node name="input">
    <transition to="approval" />
  </task-node>

  <task-node name="approval" >
    <transition name="yes" to="account" />
    <transition name="no"  to="End" />
  </task-node>

  <task-node name="account" >
    <transition to="End" />
  </task-node>

  <end-state name="End" />

</process-definition>
```

JPDLで表現できるフロー

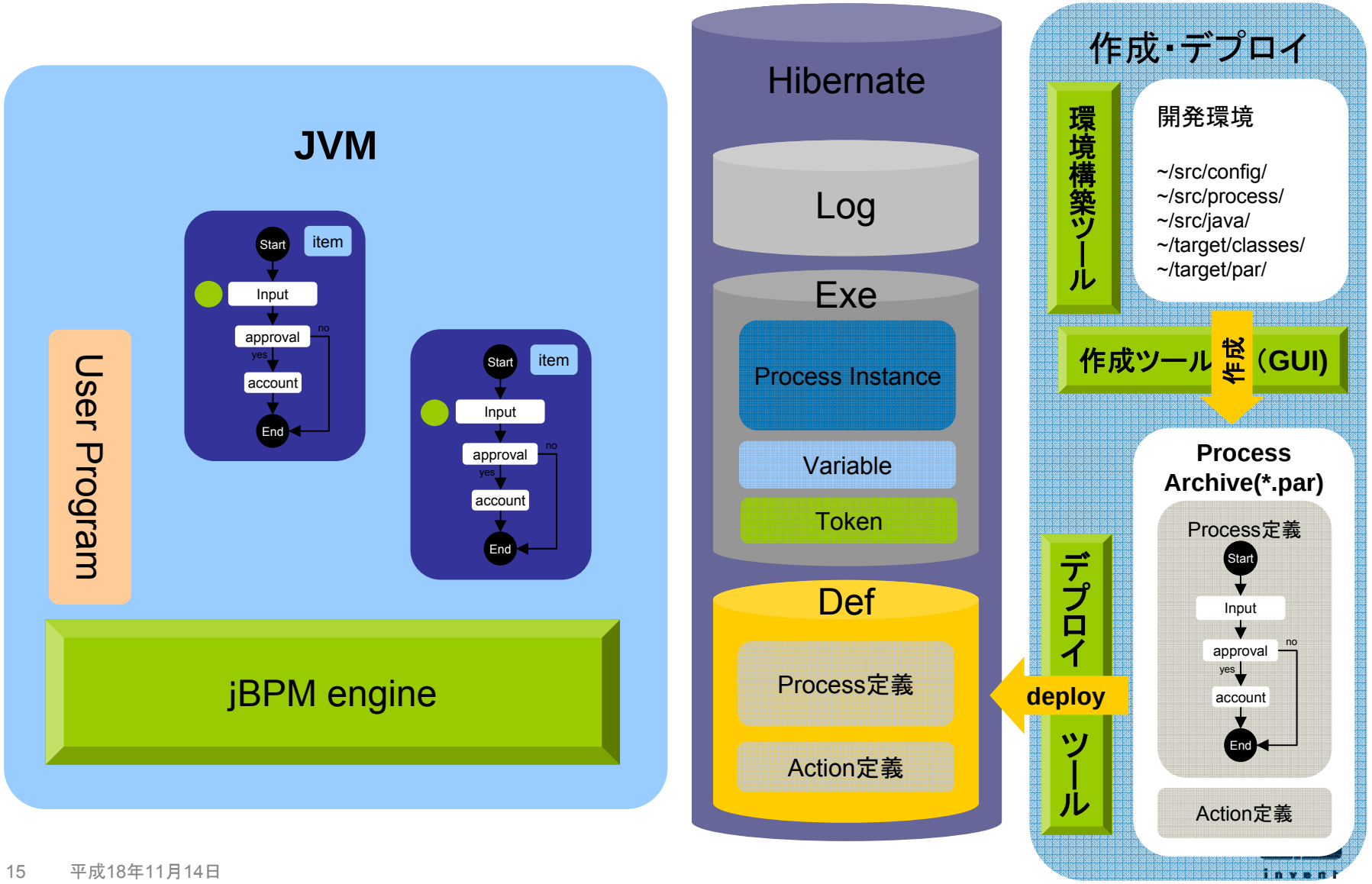
- State
- Task-node
- State-node
- Node
- Decision
- ForkとJoin
- Subprocess



ソフトウェアスタック

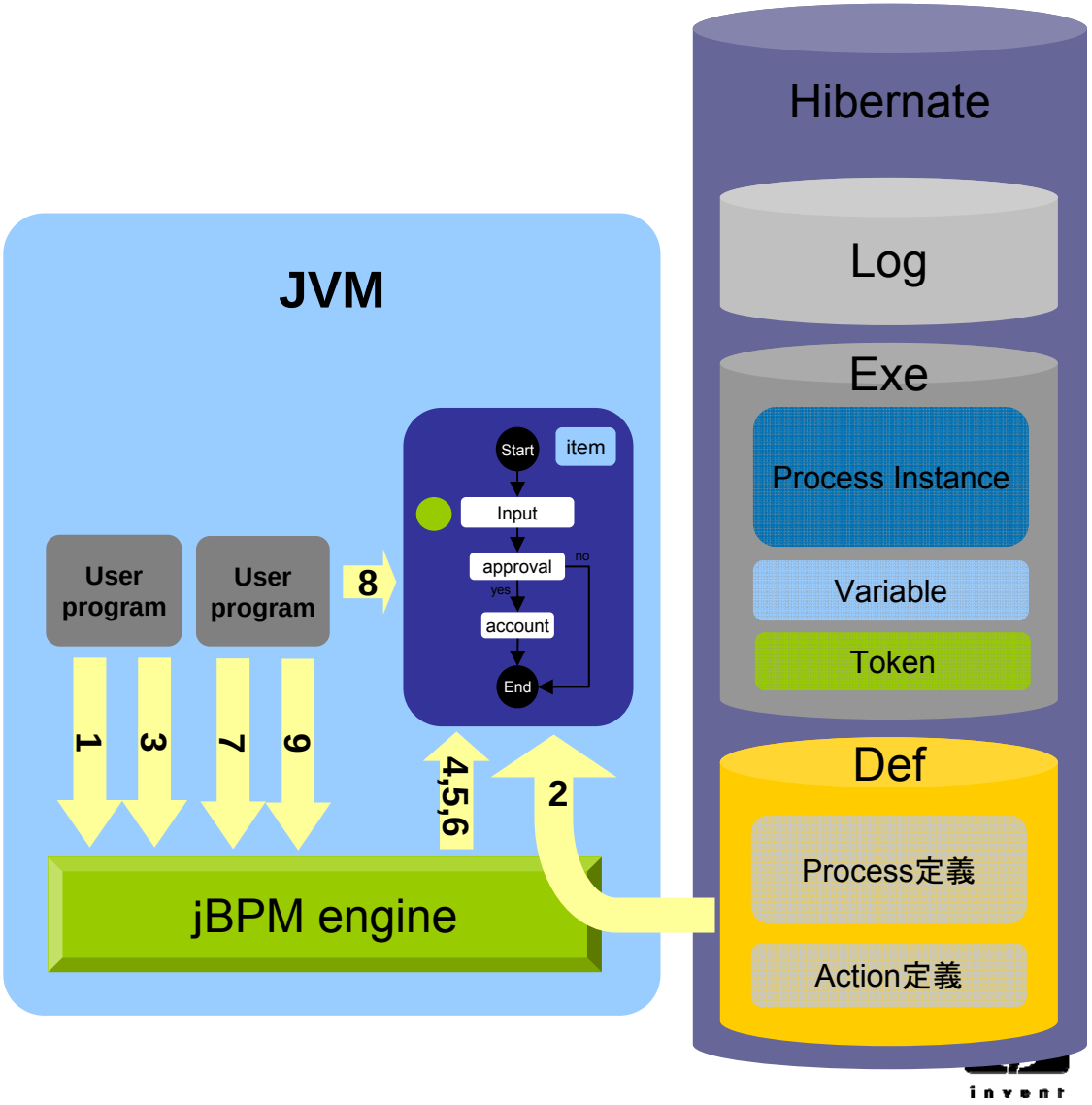

プロセスアーカイブ
Process定義
Start
Input
approval
no
yes
account
End
Action定義
開発環境
~/src/config/
~/src/process/
~/src/java/
~/target/classes/
~/target/par/
User Program
jBPM 開発ツール（GUI）
jBPM デプロイツール
Database
jBPM engine
Hibernate
J2SE

システム全体図


JVM
User Program
Start
item
Input
approval
no
yes
account
End
jBPM engine
Hibernate
Log
Exe
Process Instance
Variable
Token
Def
Process定義
Action定義
作成・デプロイ
環境構築ツール
開発環境
~/src/config/
~/src/process/
~/src/java/
~/target/classes/
~/target/par/
作成ツール
作成
(GUI)
Process Archive(*.par)
Process定義
Action定義
デプロイツール
deploy

プロセスの生成と動作

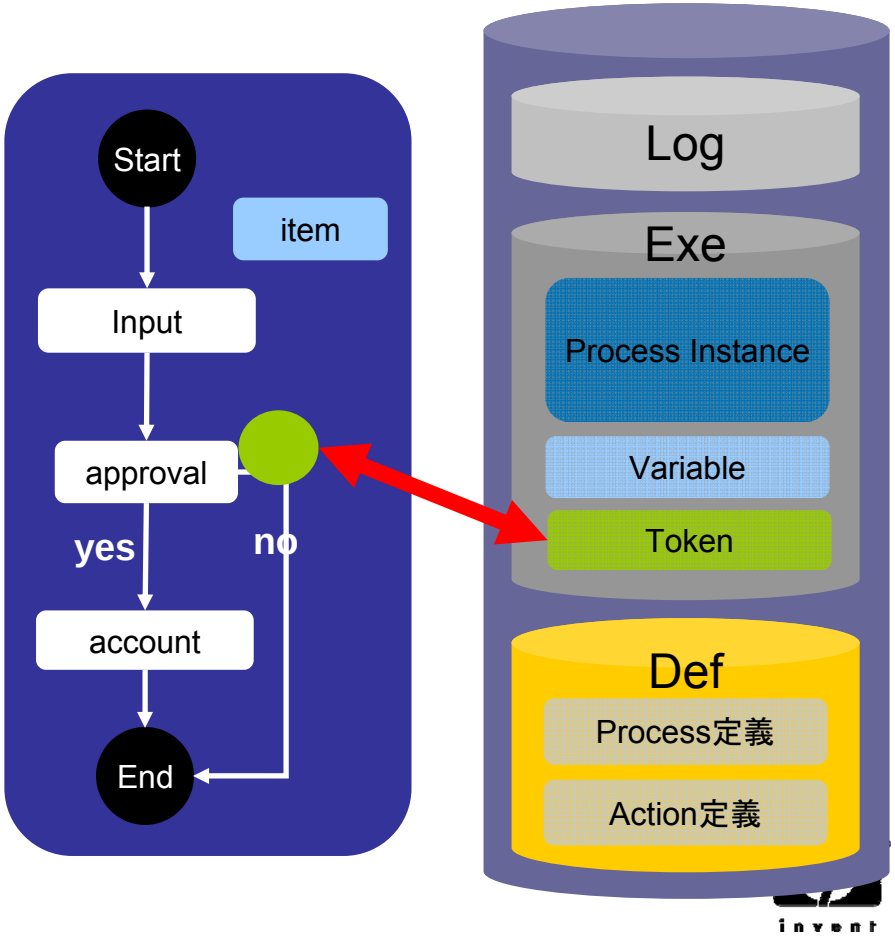
1. ユーザプログラムがjBPMエンジンにプロセスの生成を依頼
2. jBPMエンジンがプロセス定義を参照しプロセスを生成
3. ユーザプログラムがエンジンにプロセスの実行を依頼
4. jBPMエンジンがプロセスの実行を開始
5. プロセスを順次実行
6. 必要があればユーザプログラムに制御を渡すためにプロセスを待ち状態にする
7. ユーザプログラムが待っているプロセスのリストを取得
8. 取得したプロセスに対して処理を行う
9. プロセスの続行をエンジンに依頼



プロセスの動作概要

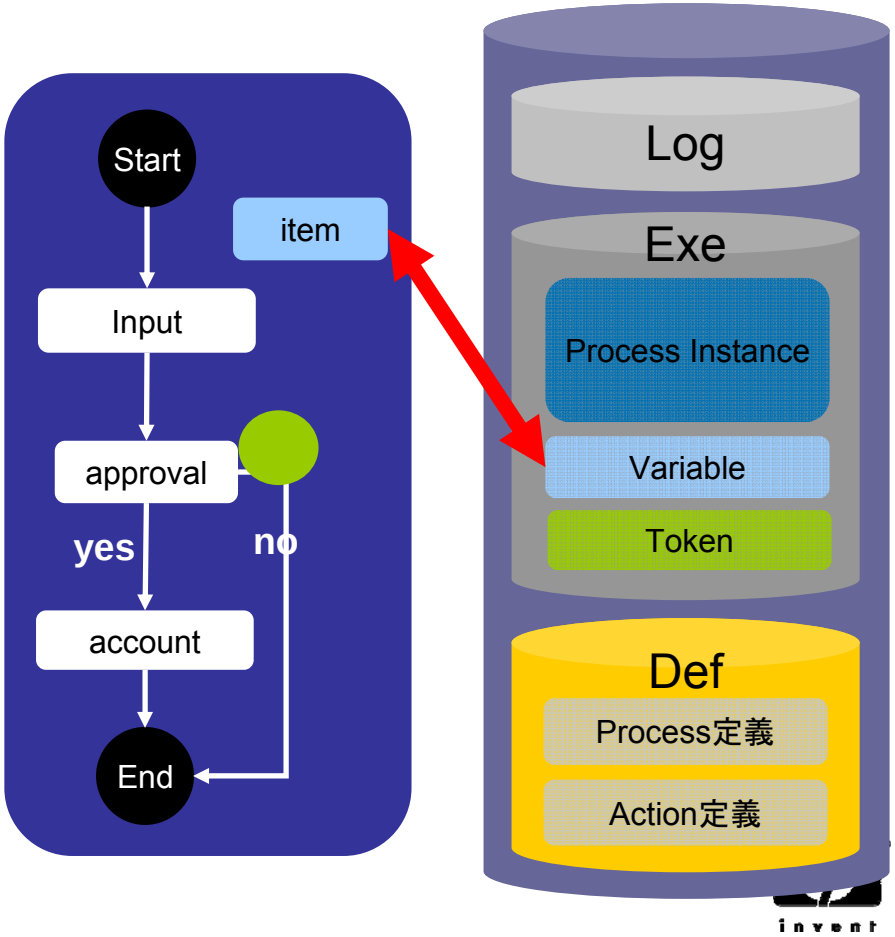
Token (トークン)

- プロセスの遷移を表現する
- プロセスの現在の位置を保持するオブジェクト
- 以下の情報を保持
  - プロセスインスタンスへのリンク情報
  - ノード(位置)情報



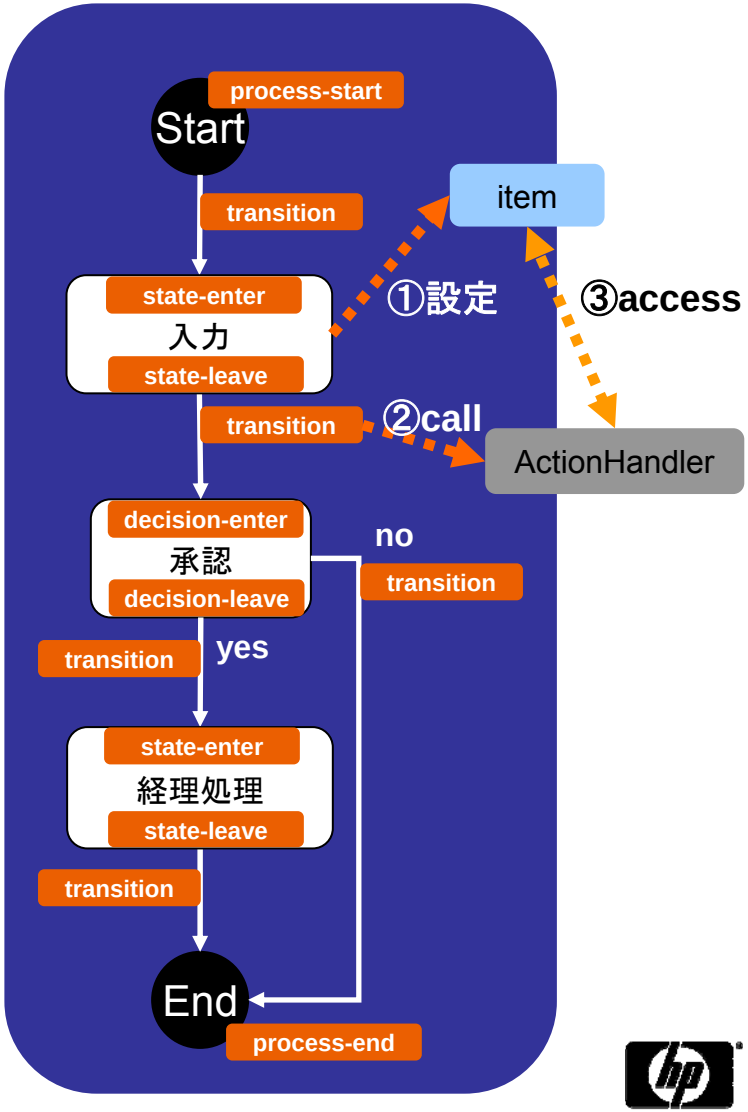
プロセス変数(Variable)

- 各プロセスインスタンスが保持する変数
- データベースに保持される
- ユーザー定義のデータを使用する場合は独自にシリアライザを作成する必要がある



アクション

- プロセスのステートが遷移するときに自動的に呼び出されるユーザ処理
- アクションは、ActionHandlerインターフェースのexecute()メソッドで実装
- JPDL定義ファイルでアクションを定義
- アクションを設定できる場所
  - プロセスの開始と終了
  - ステートに入ったとき(ENTER)
  - ステートから出る時(LEAVE)
  - ステートが変わる時(TRANSITION)
- execute()メソッド内から、同メソッドの引数であるActionContextを用いてプロセス変数にアクセス可能
  - プロセス変数経由でプロセスを呼び出したプログラムと情報の交換が可能
- 使用例
  - 外部システムとの連携



プロセスとクライアントプログラム

クライアントプログラム

プロセス生成の実行

プロセスの開始

社員が自分自身に関連するプロセスを取得

承認する商品を入力

商品を“item”変数に設定

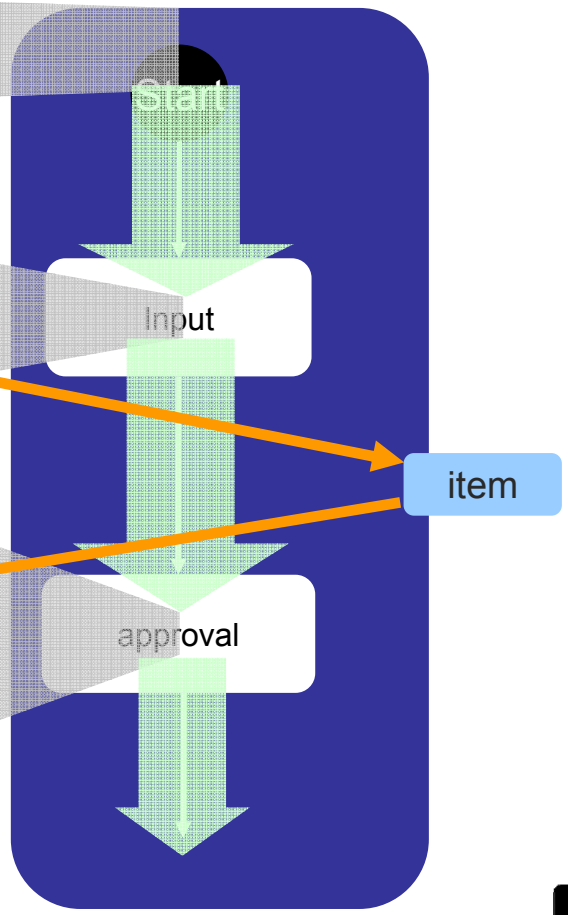
次に進む

上司が自分自身に関連するプロセスを取得

“item”変数から商品を取得

承認するかしないかを判断

次に進む



jBPMアプリケーション

購買アプリケーション

- 備品を購入するための申請・承認システム
  - 登場人物：社員（申請者）、上司（承認者）

- 社員
  - 購買申請処理
    - 商品と値段の入力
    - 承認する上司の選択
    - 承認が緊急であるかどうかを入力
      - 緊急の場合は、承認者にメールが自動送信される
  - 申請履歴の参照
    - 承認結果を参照
- 上司
  - 申請の承認処理
    - 承認結果は、申請者にメールで知らされる

購買アプリケーションのユースケース

<社員：購買申請>

Step1：システムにログイン

Step2：購買申請メニューへ進む

Step3：購買品目情報を入力し、次の画面へ

- 品名、価格

Step4：次の情報を選択し、購買申請を送信

- 承認依頼する上司
  - 外部システムにアクセスし、価格に応じて選択可能な上司の一覧が表示される
- 緊急かどうか
  - 緊急な場合は上司にメールが送信される

Step5：申請確認画面が表示される

- 申請ID、品名、価格、上司名

Step6：ログアウト

<上司：承認処理>

Step1：システムにログイン

Step2：承認処理メニューへ進む

Step3：申請を承認するか否かを選択し、確定

- 承認結果が社員にメール送信される

Step4：ログアウト

<社員：申請履歴参照>

Step1：システムにログイン

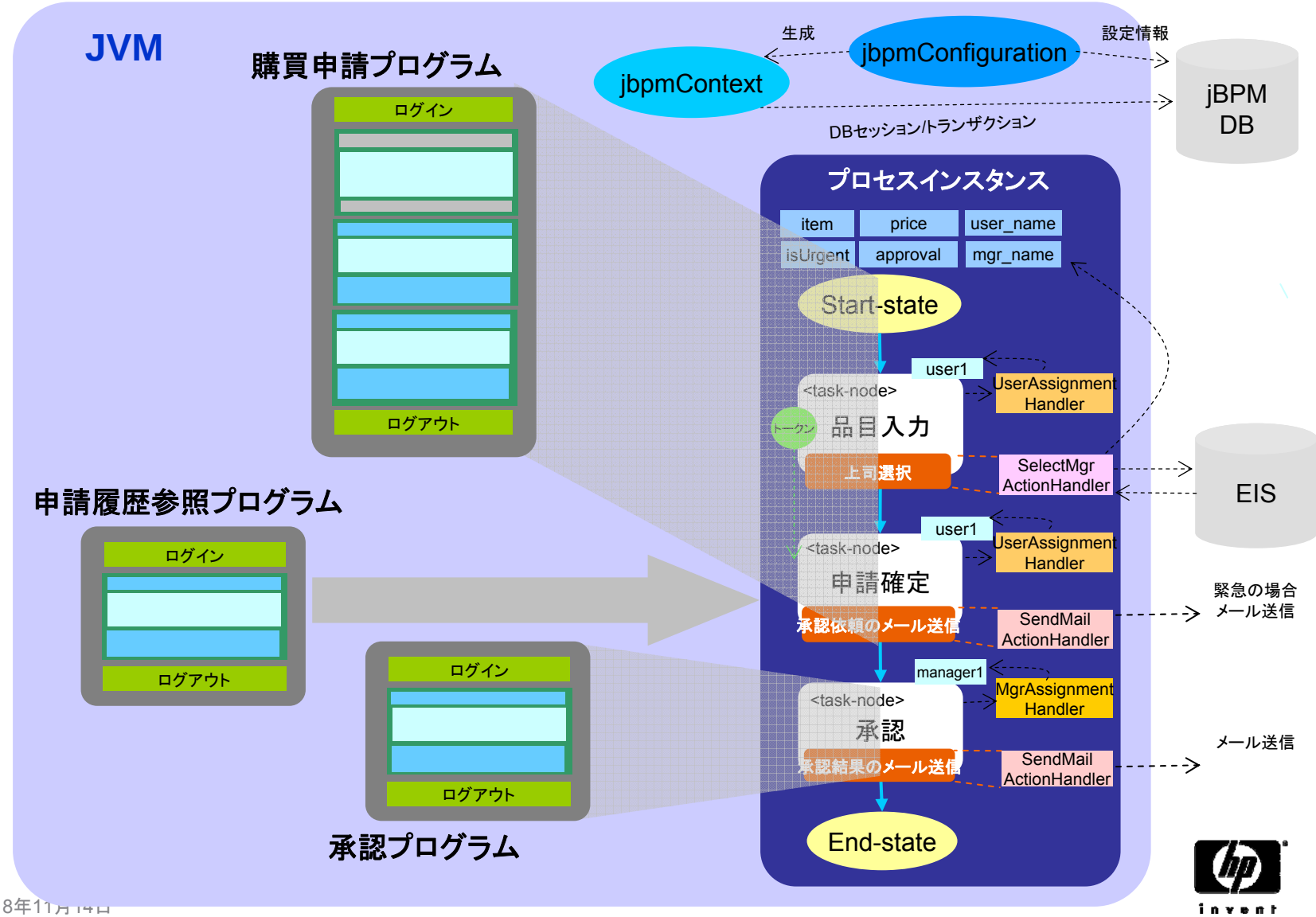
Step2：購買申請履歴メニューへ進む

Step3：過去の申請IDを入力すると、申請のステータスが表示される

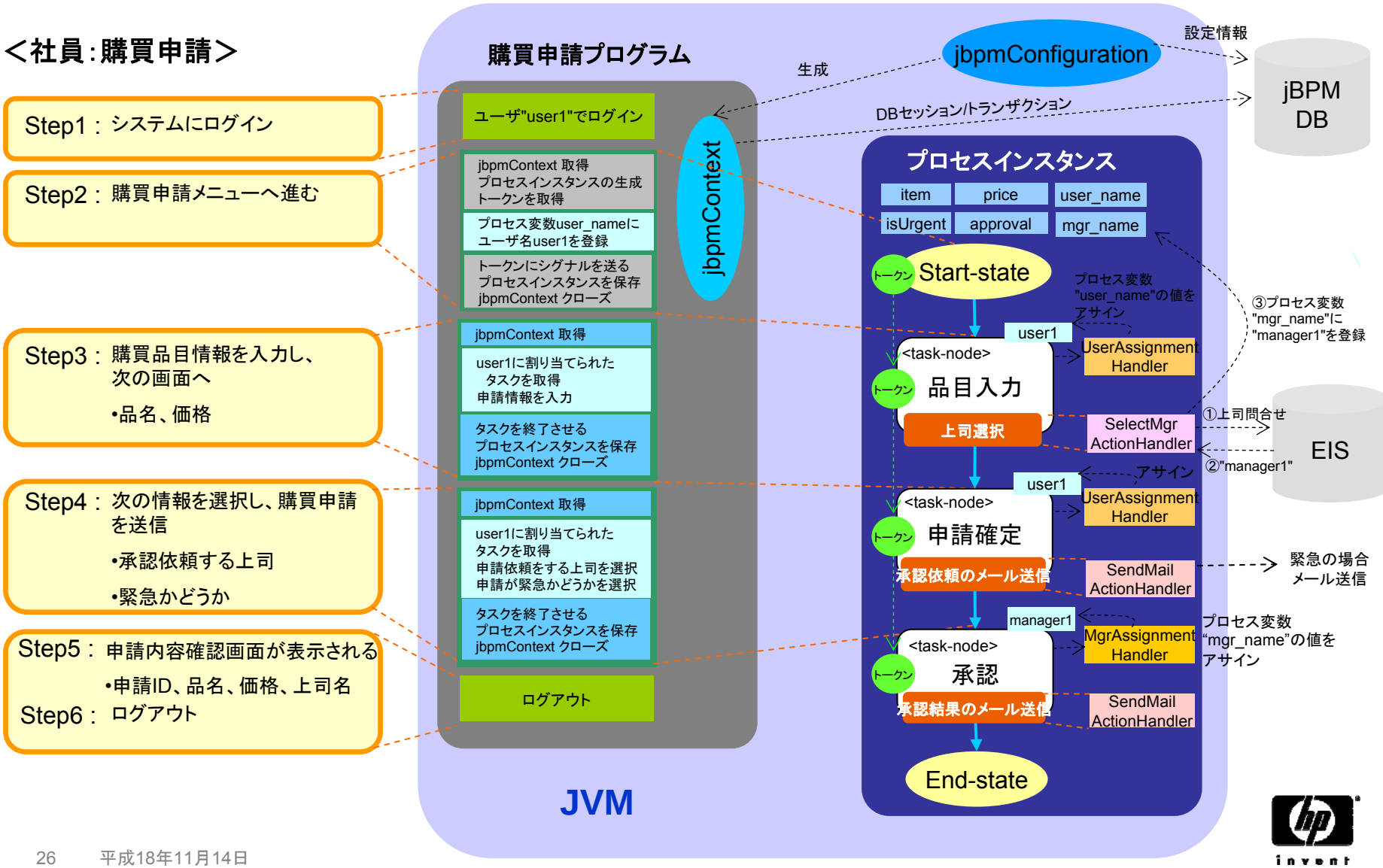
- 申請ID(プロセスID)
- 品目、価格
- ステータス(承認済、未承認、却下)

Step4：ログアウト

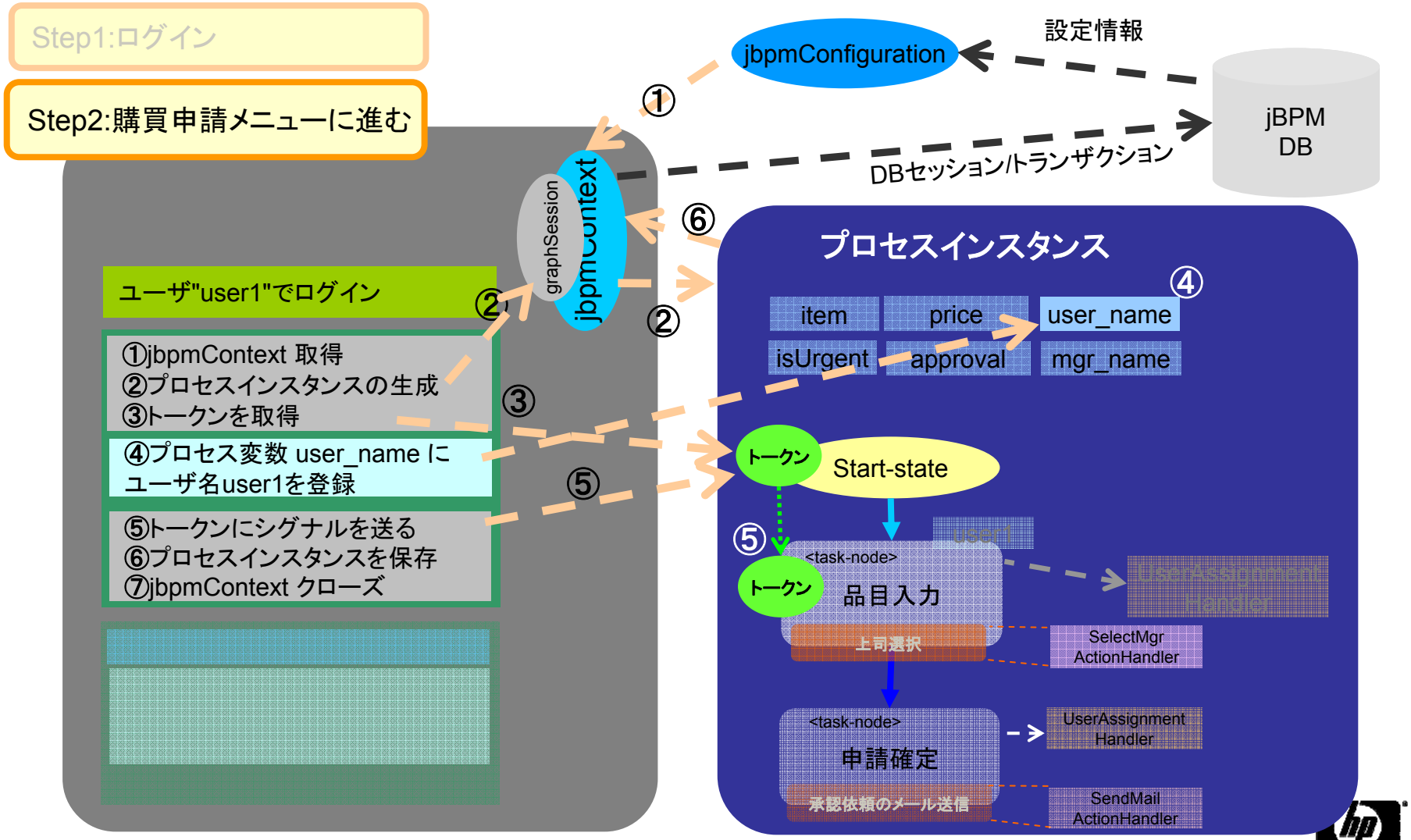
購買アプリケーション全体図



購買申請プログラム

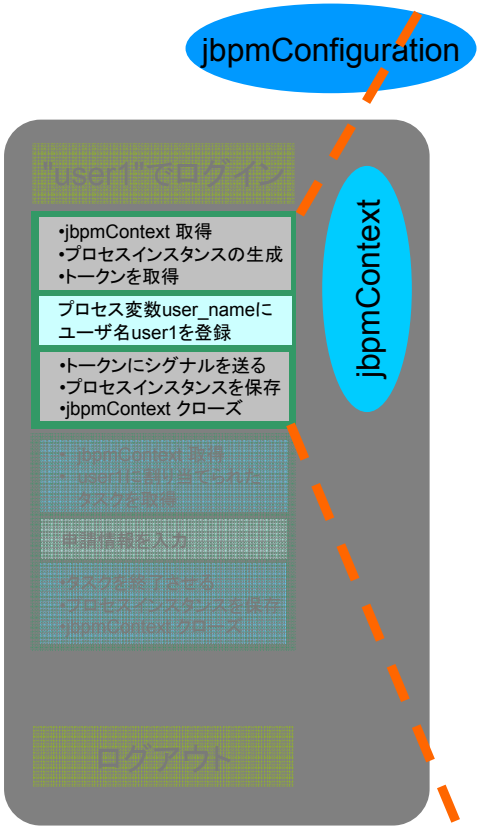

＜社員：購買申請＞
Step1： システムにログイン
Step2： 購買申請メニューへ進む
Step3： 購買品目情報を入力し、次の画面へ
•品名、価格
Step4： 次の情報を選択し、購買申請を送信
•承認依頼する上司
•緊急かどうか
Step5： 申請内容確認画面が表示される
•申請ID、品名、価格、上司名
Step6： ログアウト
購買申請プログラム
ユーザ"user1"でログイン
jbpmContext 取得
プロセスインスタンスの生成
トークンを取得
プロセス変数user_nameに
ユーザ名user1を登録
トークンにシグナルを送る
プロセスインスタンスを保存
jbpmContext クローズ
jbpmContext 取得
user1に割り当てられた
タスクを取得
申請情報を入力
タスクを終了させる
プロセスインスタンスを保存
jbpmContext クローズ
jbpmContext 取得
user1に割り当てられた
タスクを取得
申請依頼をする上司を選択
申請が緊急かどうかを選択
タスクを終了させる
プロセスインスタンスを保存
jbpmContext クローズ
ログアウト
jbpmContext
JVM
生成
jbpmConfiguration
設定情報
DBセッション/トランザクション
jBPM
DB
プロセスインスタンス
item
price
user_name
isUrgent
approval
mgr_name
トークン
Start-state
プロセス変数
"user_name"の値を
アサイン
user1
<task-node>
品目入力
UserAssignment
Handler
上司選択
SelectMgr
ActionHandler
③プロセス変数
"mgr_name"に
"manager1"を登録
①上司問合せ
②"manager1"
EIS
アサイン
user1
<task-node>
申請確定
UserAssignment
Handler
承認依頼のメール送信
SendMail
ActionHandler
緊急の場合
メール送信
manager1
<task-node>
承認
MgrAssignment
Handler
プロセス変数
“mgr_name”の値を
アサイン
承認結果のメール送信
SendMail
ActionHandler
End-state

プロセスインスタンスの生成



プロセスインスタンスの生成（コード）

Step2:購買申請メニューへ

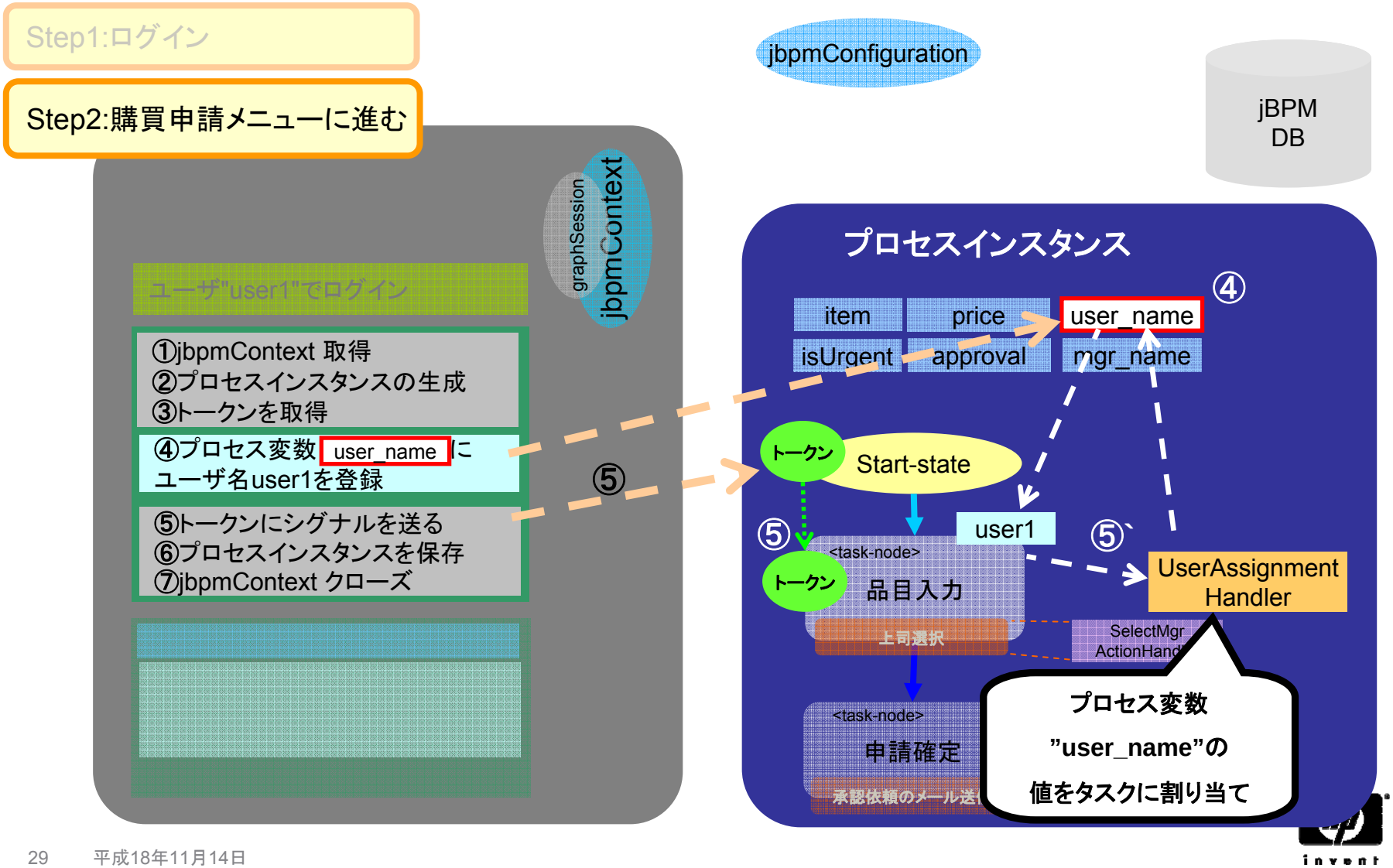



```
// jBPMContextの取得
JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
GraphSession graphSession =  jbpmContext.getGraphSession();

try {
  // プロセスインスタンスの生成
  ProcessDefinition pd =
    graphSession.findLatestProcessDefinition(procName);
  ProcessInstance pi = new ProcessInstance(pd);

  // トークンを取得
  Token token = pi.getRootToken();
```

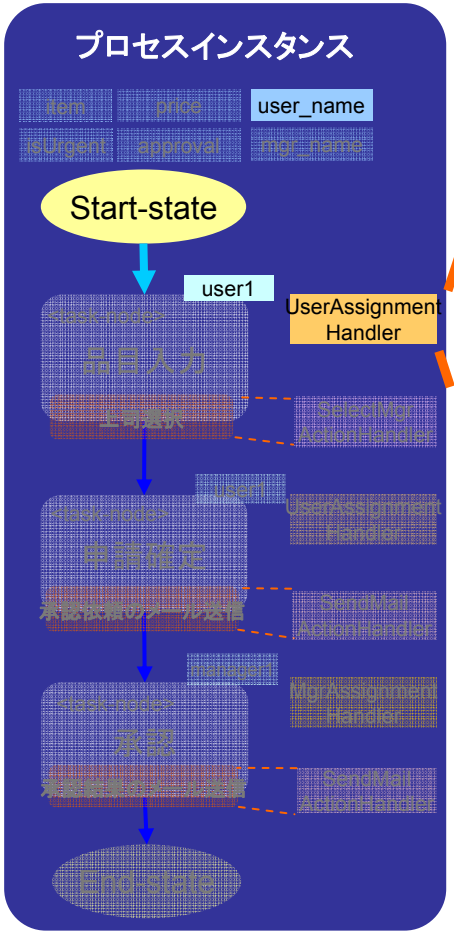
```
  // プロセス変数"user_name"に、ユーザ名"user1"を登録
  pi.getContextInstance().setVariable("user_name", getUserName());
```

```
  // トークンにシグナルを送る
  token.signal();

  // プロセスインスタンスを保存
  jbpmContext.save(pi);

} finally {
  jbpmContext.close(); // jBPMContextのクローズ
}
```

taskへのユーザー登録



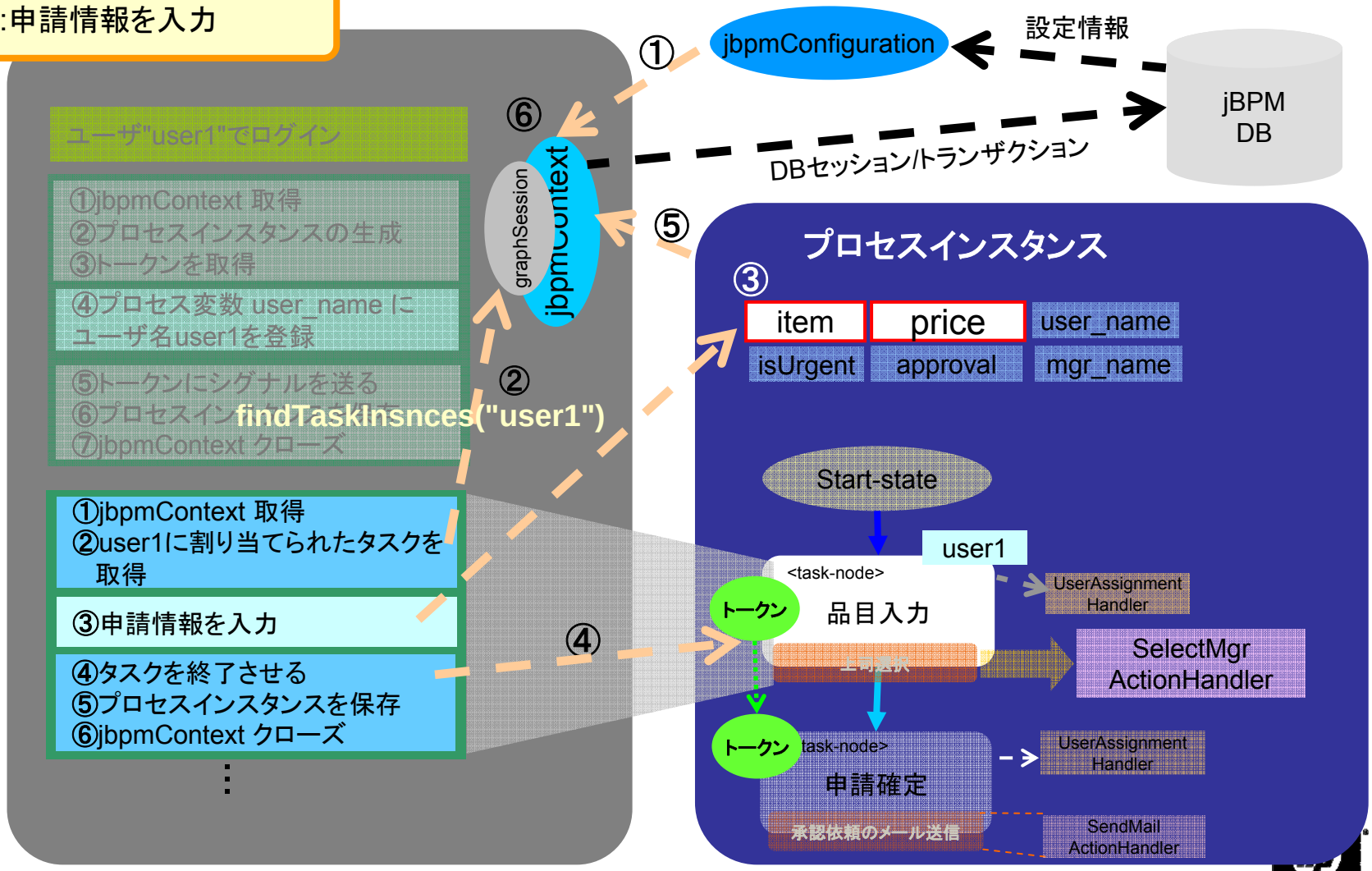
taskへのユーザー登録（UserAssignmentHandler）




```
import org.jbpm.graph.exe.ExecutionContext;
import org.jbpm.taskmgmt.def.AssignmentHandler;
import org.jbpm.taskmgmt.exe.Assignable;

public class UserAssignmentHandler implements AssignmentHandler
{
    public void assign(Assignable assignable,
      ExecutionContext executionContext) {

        // プロセス変数"user_name"からアサイン対象のユーザ名を取得
        String user =
          (String)executionContext.getVariable("user_name");
        // アサインする
        assignable.setActorId(user);

    }
}
```

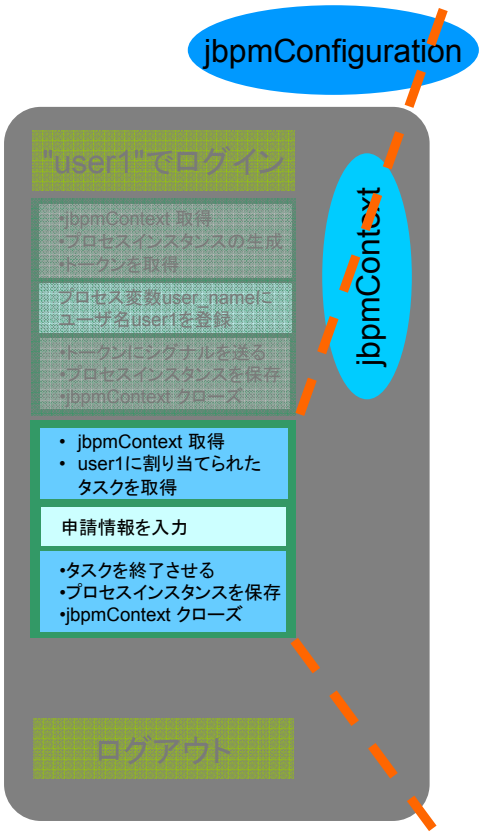
タスクの取得

Step3:申請情報を入力



タスクの取得(コード)

Step3:申請情報を入力

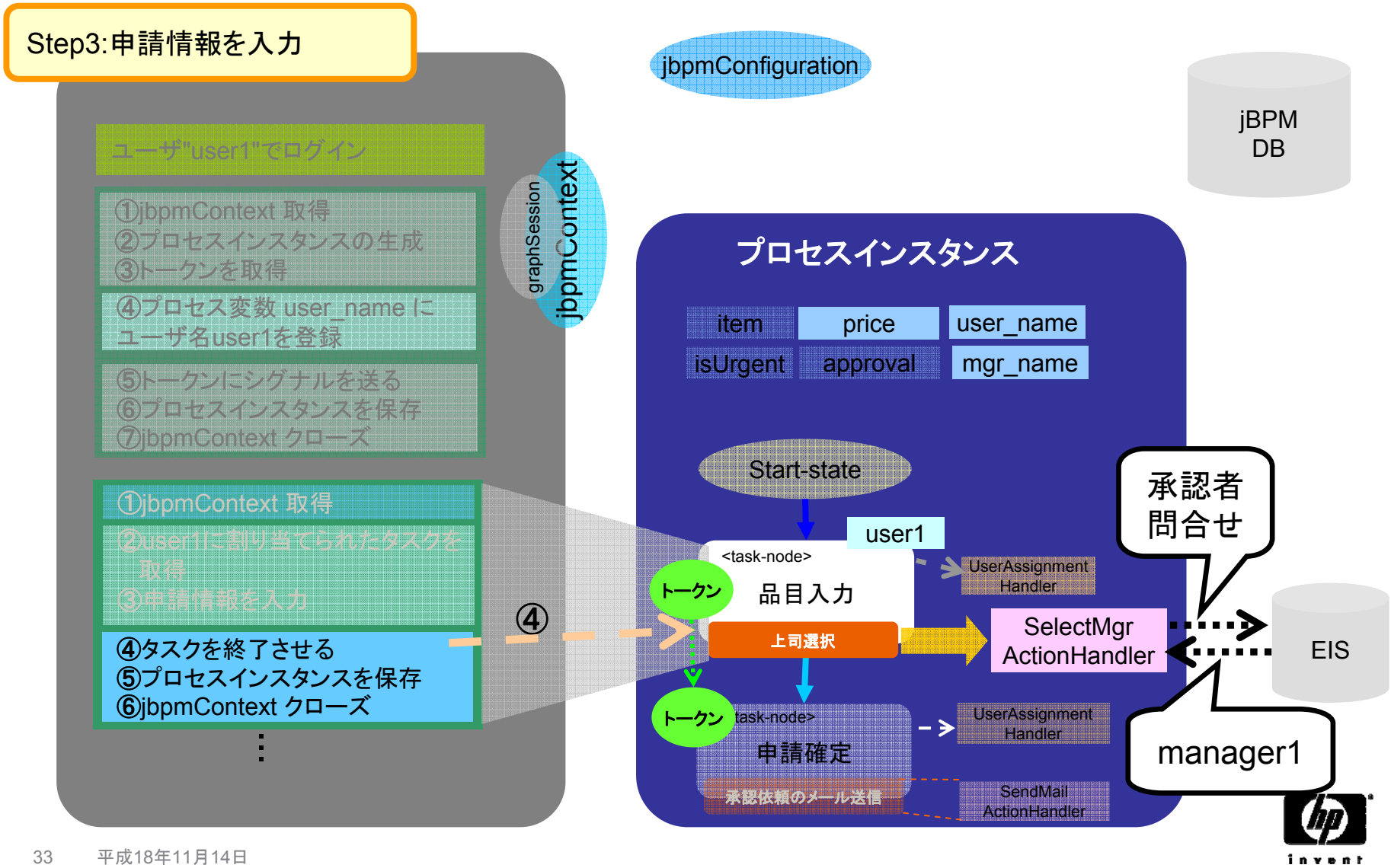



```
// jbpmContextの取得
JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
TaskMgmtSession taskMgmtSession = jbpmContext.getTaskMgmtSession();

try {
  // user1に割り当てられた最初の"apply item task"タスクを処理
  Iterator it =
     taskMgmtSession.findTaskInstances(getUserName()).iterator();
  TaskInstance ti = null;
  for (; it.hasNext(); ) {
    ti = (TaskInstance)it.next();
    if (ti.getName().equalsIgnoreCase("apply item task")) {
```

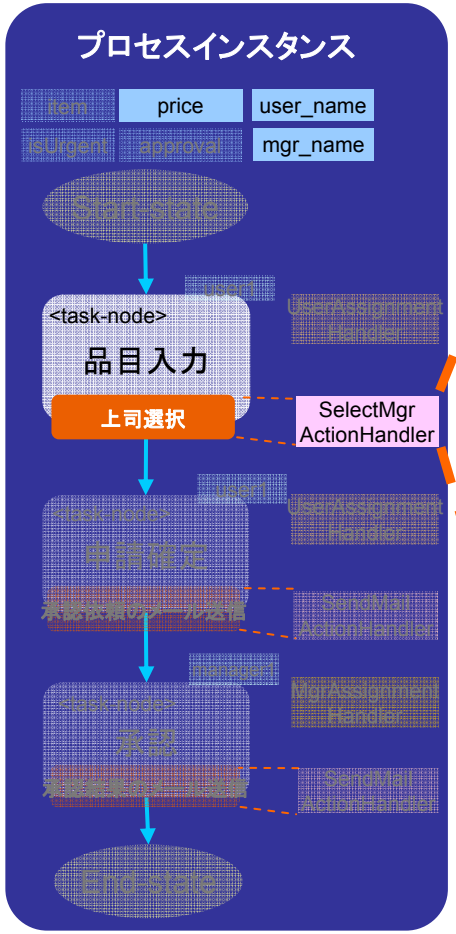
```
      // 申請情報を入力
      ti.setVariable("item", getItem());  // 品名
      ti.setVariable("price", getPrice());  // 価格
      ti.setVariable("applicant", getUserName()); // ユーザ名"user1"
      break;
    }
  }
```

```
  ti.end();              // タスクを終了させる
  jbpmContext.save(pi); // プロセスインスタンスを保存

} finally {
  jbpmContext.close();  // jbpmContextのクローズ
}
```

アクションハンドラーの動作



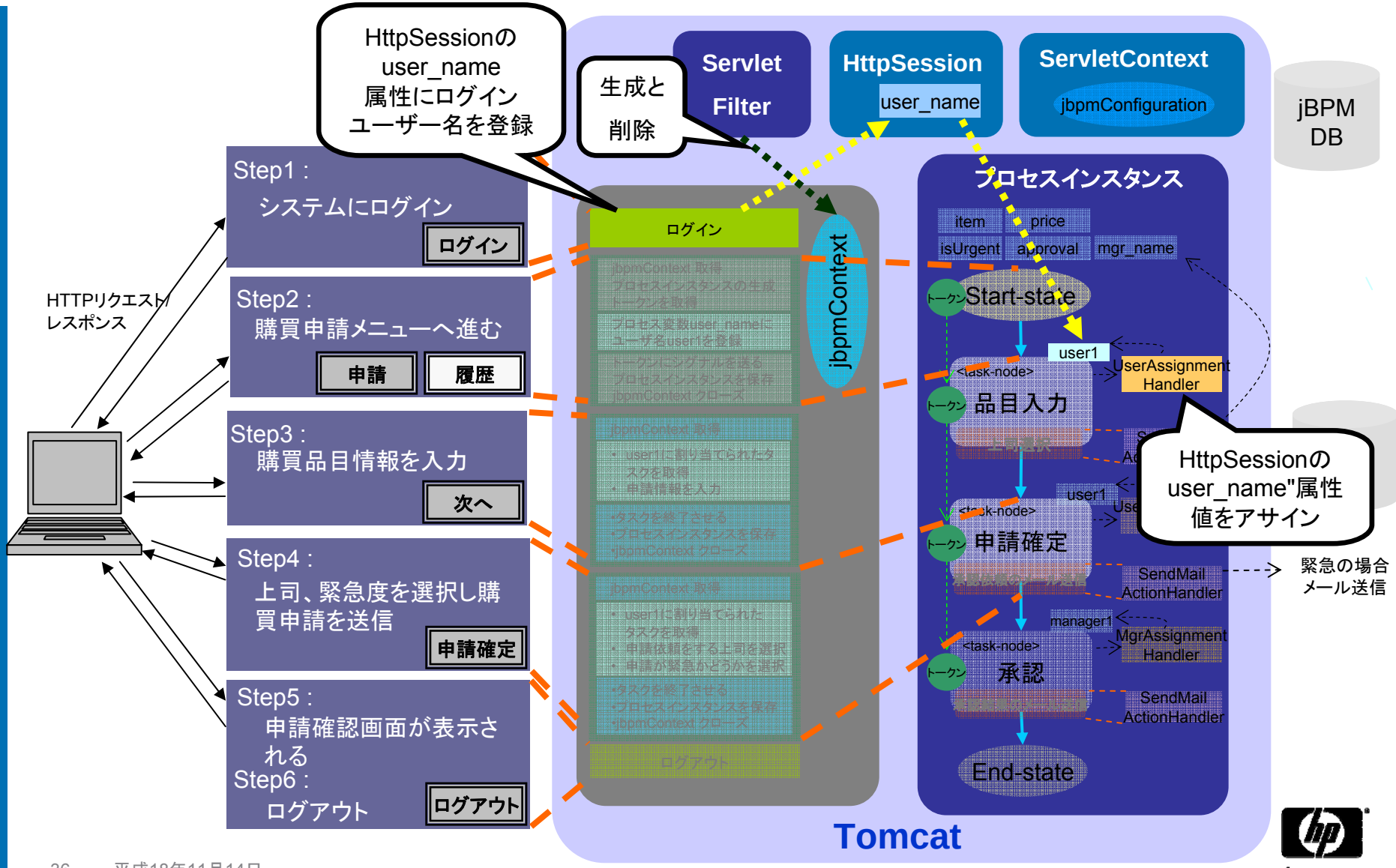
アクションハンドラーのコード（SelectMgrActionHandler）




```
import org.jbpm.graph.def.ActionHandler;
import org.jbpm.graph.exe.ExecutionContext;

public class SelectMgrActionHandler implements ActionHandler {
    public void execute(ExecutionContext executionContext) {

        // プロセス変数から、申請者と価格の情報を取得
        String applicant =
          (String)executionContext.getVariable("user_name");
        String price =
          (String)executionContext.getVariable("price");

        // 承認対象上司を取得
        //   外部データベース、ディレクトリサーバにアクセスして、
        //   申請者と価格に応じた承認対象の上司を決定する....
        String manager = getManager(applicant, price);

        // プロセス変数"mgr_name"に承認対象上司を登録
        executionContext.setVariable("mgr_name", manager);

    }
}
```

jBPM webアプリケーション

Webアプリケーション（購買申請プログラム）

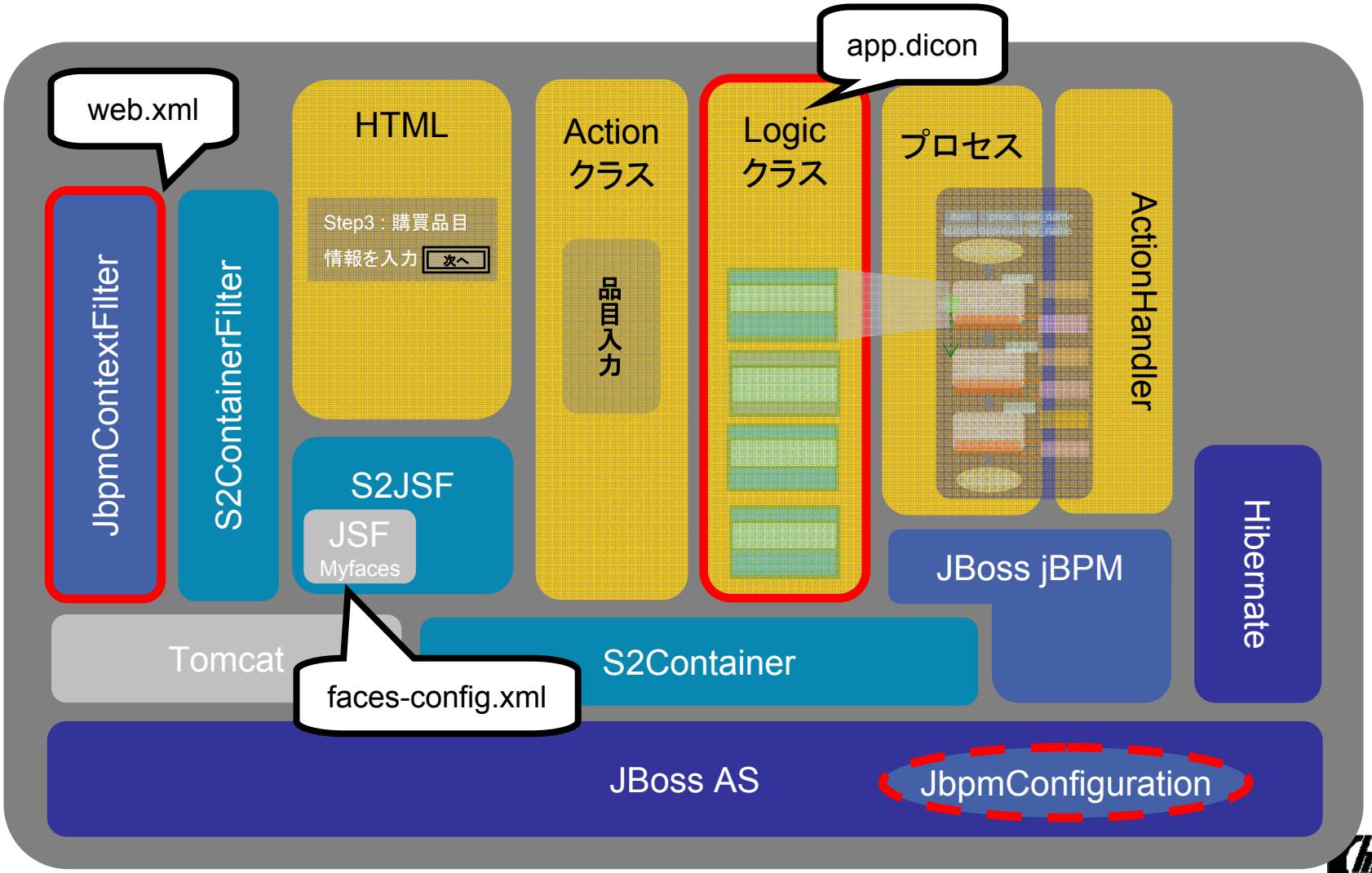


Seasar2との連携

Seasar2との連携

- 購買アプリケーションのWebシステムをJBossとSeasar2で構築
  - J2EEサーバ ：JBoss AS 4.0.5
  - DIxAOP コンテナ：S2Container 2.3.10
  - プレゼンテーション・フレームワーク：S2JSF 1.0.18
  - ビジネスロジックエンジン：JBoss jBPM 3.1.2
    - O/R Mapper : Hibernate 3.1

ソフトウェアスタック


app.dicon
web.xml
HTML
Step3：購買品目
情報を入力
次へ
Action
クラス
品目入力
Logic
クラス
プロセス
ActionHandler
JbpmContextFilter
S2ContainerFilter
S2JSF
JSF
Myfaces
JBoss jBPM
Hibernate
Tomcat
S2Container
faces-config.xml
JBoss AS
JbpmConfiguration

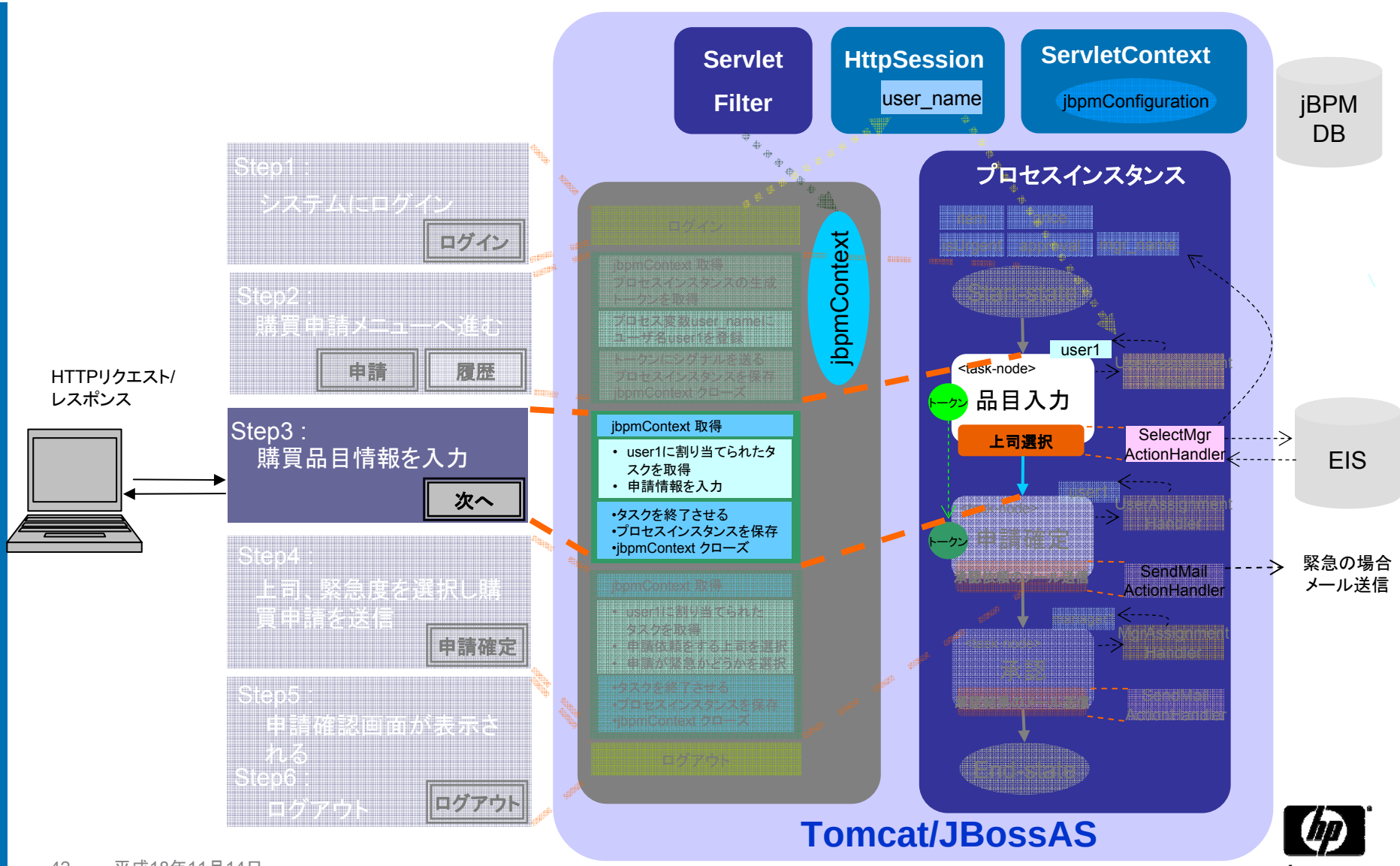
連携する上で重要な設定

- web.xml
  - Servlet Filter
    - jBPMのjbpmContextを設定するフィルターを設定
      - **org.jbpm.web.JbpmContextFilter**
- app.dicon
  - DIの設定
    - jBPMのAPIをコールするロジッククラス
- jbpm.sarをJBossにデプロイ
  - jBPMをJBossと連携させるために必要
  - jbpmContext取得用JBoss JNDI サービス

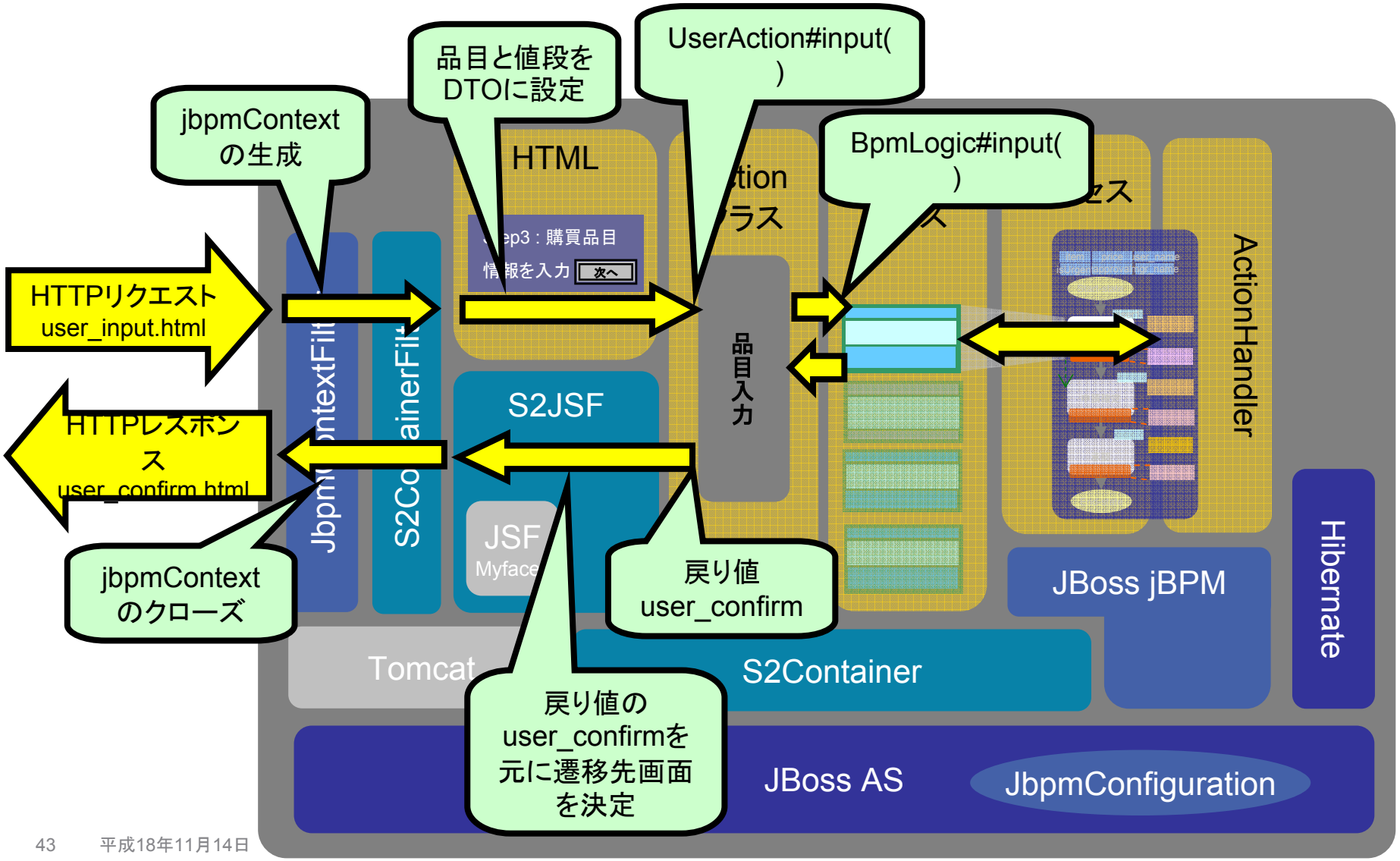
基本的な設定

- faces-config.xml
  - 画面遷移の設定
- app.dicon
  - DIの設定
    - アクションクラス
    - DTO

Step3：購買品目情報を入力



システム制御フロー


品目と値段を
DTOに設定
UserAction#input(
)
jbpmContext
の生成
BpmLogic#input(
)
HTML
HTTPリクエスト
user_input.html
S2JSF
品目入力
ActionHandler
HTTPレスポン
ス
user_confirm.html
JSF
Myface
jbpmContext
のクローズ
戻り値
user_confirm
JBoss jBPM
Hibernate
Tomcat
S2Container
戻り値の
user_confirmを
元に遷移先画面
を決定
JBoss AS
JbpmConfiguration

JbpmContextFilter

- jbpmContextの生成とクローズを実施
- JBoss jBPM に付属のServlet Filter
- クライアントからのHTTPリクエスト受付時にJbpmContextを生成
  - 生成されたJbpmContextはスレッドに割り当てられる(ThreaLocal)
- クライアントにHTTPレスポンスを返す前に、JbpmContextをクローズ

web.xml

```
...
<filter>
  <filter-name>JbpmContextFilter</filter-name>
  <filter-class>org.jbpm.web.JbpmContextFilter
  </filter-class>
</filter>
<filter-mapping>
  <filter-name>JbpmContextFilter</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
  ...
```

JbpmContextFilter.java

```
public class JbpmContextFilter implements Filter, Serializable {
...
  public void doFilter(ServletRequest servletRequest,
    ServletResponse servletResponse, FilterChain filterChain)
    throws IOException, ServletException {
    JbpmContext jbpmContext = getJbpmConfiguration().createJbpmContext(jbpmContextName);
    try {
       if (isAuthenticationEnabled) { jbpmContext.setActorId(actorId); }
       filterChain.doFilter(servletRequest, servletResponse);
     } finally {
       jbpmContext.close();
    }
  }
```

HTTPリクエスト：user_input.html

```
<h2>Enter item to apply!</h2>
<span m:inject="h:messages" m:globalOnly="false                    rue"/>
<form>
  <table border="0">
    <tr>
      <td>Item </td>
      <td><input type="text" m:value="#{applyDto.item}"/></td>
    </tr>
    <tr>
      <td>Price </td>
      <td><input type="text" m:value="#{applyDto.price}"/></td>
      </tr>
      </table>
  <input type="submit" value="Next" m:action="#{userAction.input}"/>
</form>
```

DTOにitemとpriceの値を設定

アクションクラスの呼び出し

アクションクラス：UserAction#input()

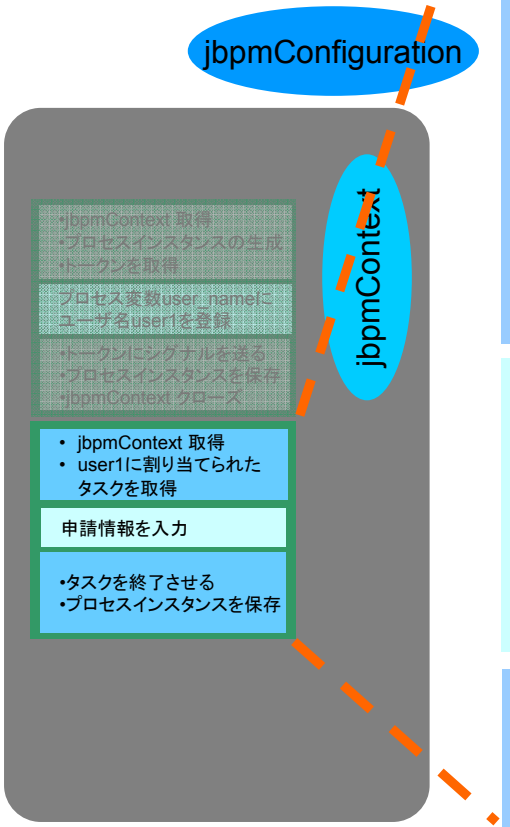
```
public class UserActionImpl implements UserAction {
  ...
  private BpmLogic bpmLogic;
  ...
  public void setBpmLogic(BpmLogic bpmLogic) {
    this.bpmLogic = bpmLogic;
  }
  ...
  public String input() {
    bpmLogic.input();
    return "user_confirm";
  }
  ...
}
```

ロジッククラスのDI

- ロジッククラスの呼び出し
- 次の遷移先の指定

ロジッククラス：BpmLogic#input()

Step3:申請情報を入力




```
public class BpmLogicImpl implements BpmLogic {

public BpmLogicImpl() { // コンストラクタ
    // jbpmContext取得
    this.jbpmContext = JbpmContext.getCurrentJbpmContext();
    this.taskMgmtSession = jbpmContext.getTaskMgmtSession();
  }
  ...
  public void input() {
    // user1に割り当てられた最初の"apply item task"タスクを処理
    String username = userDto.getUsername();
    //以下は同じコードなので省略…
    for (…) {
      if (ti.getName().equalsIgnoreCase("apply item task")) {

        // 申請情報を取得
        ti.setVariable("item", applyDto.getItem());   // 品名
        ti.setVariable("price", applyDto.getPrice()); // 価格
        ti.setVariable("applicant", username);  // ユーザ名
        break;
      }
    }

    ti.end();                 // タスクを終了させる
    jbpmContext.save(pi); // プロセスインスタンスを保存
    // jbpmContext のクローズ処理は、ここでは不要
  }
}
```

画面遷移：faces-config.xml

- Webアプリケーションの画面遷移はJSFの faces-config.xml で記述

```
<faces-config>

 <!- Step4 申請情報確認画面 -->
 <navigation-rule>
    <navigation-case>
      <from-outcome>user_confirm</from-outcome>
      <to-view-id>/user_confirm.html</to-view-id>
      <redirect/>
    </navigation-case>
  </navigation-rule>

  ...
</faces-config>
```

app.dicon：DIの設定

- **アクションクラス**
  - 画面遷移先を決定するためのクラス
- **DTO**
  - アクションクラスやロジッククラスからフォームに入力されたデータを利用するためのクラス
  - アプリケーションからデータベース上のデータを利用するためのクラス
- **ロジッククラス**
  - JBoss jBPMエンジンのAPIを実際にコールするクラスの設定
  - アクションクラスから呼ばれる

```
<!-- アクションクラス -->
<component class="org.seasar.framework.container.
  autoregister.FileSystemComponentAutoRegister">
  <property name="instanceDef">
  @org.seasar.framework.container.deployer.InstanceDefFactory@REQUEST
  </property>
  <initMethod name="addClassPattern">
    <arg>"com.hp.kobe.oss.jbpm31.webapp.action.impl"</arg>
    <arg>".*ActionImpl"</arg>
  </initMethod>
</component>

<!-- DTO -->
...
  <property name="instanceDef">
    @org.seasar.framework.container.deployer.InstanceDefFactory@SESSION
  </property>
  <initMethod name="addClassPattern">
    <arg>"com.hp.kobe.oss.jbpm31.webapp.dto"</arg>
    <arg>".*Dto"</arg>
  </initMethod>
...

<!-- ロジッククラス -->
...
  <property name="instanceDef">
    @org.seasar.framework.container.deployer.InstanceDefFactory@REQUEST
  </property>
  <initMethod name="addClassPattern">
    <arg>"com.hp.kobe.oss.jbpm31.webapp.logic.impl"</arg>
    <arg>".*LogicImpl"</arg>
  </initMethod>
...
```

Seasar2との連携によるメリット

- S2ContainerのDI機能を使用することで、ロジッククラスは、既存のPOJOをそのまま使用可能
- 画面作成はHTML (S2JSF)で簡単作成
- 画面遷移はfaces-config.xml (JSF)で簡単作成
- S2AOPも使用可能

まとめ

- JBoss jBPMとは？
- JBoss jBPM システム概要
- プロセスの動作概要
- jBPMアプリケーション
- jBPM webアプリケーション
- Seasar2との連携

付録

参照資料

JBoss jBPMのホームページ

http://www.jbpm.org/index.html

jBPMのドキュメントのページ

http://www.jbpm.org/docs.html

jBPMのユーザフォーラム

http://sourceforge.net/forum/forum.php?forum_id=240085

jBPMのWiki

http://wiki.jboss.org/wiki/Wiki.jsp?page=JbpmWiki

Seasarプロジェクトのホームページ

http://www.seasar.org/index.html

S2Containerプロダクトのページ

http://s2container.seasar.org/ja/

S2JSFプロダクトのページ

http://s2jsf.seasar.org/ja/

弊社URL

JBossのページ

http://www.hp.com/jp/jboss/

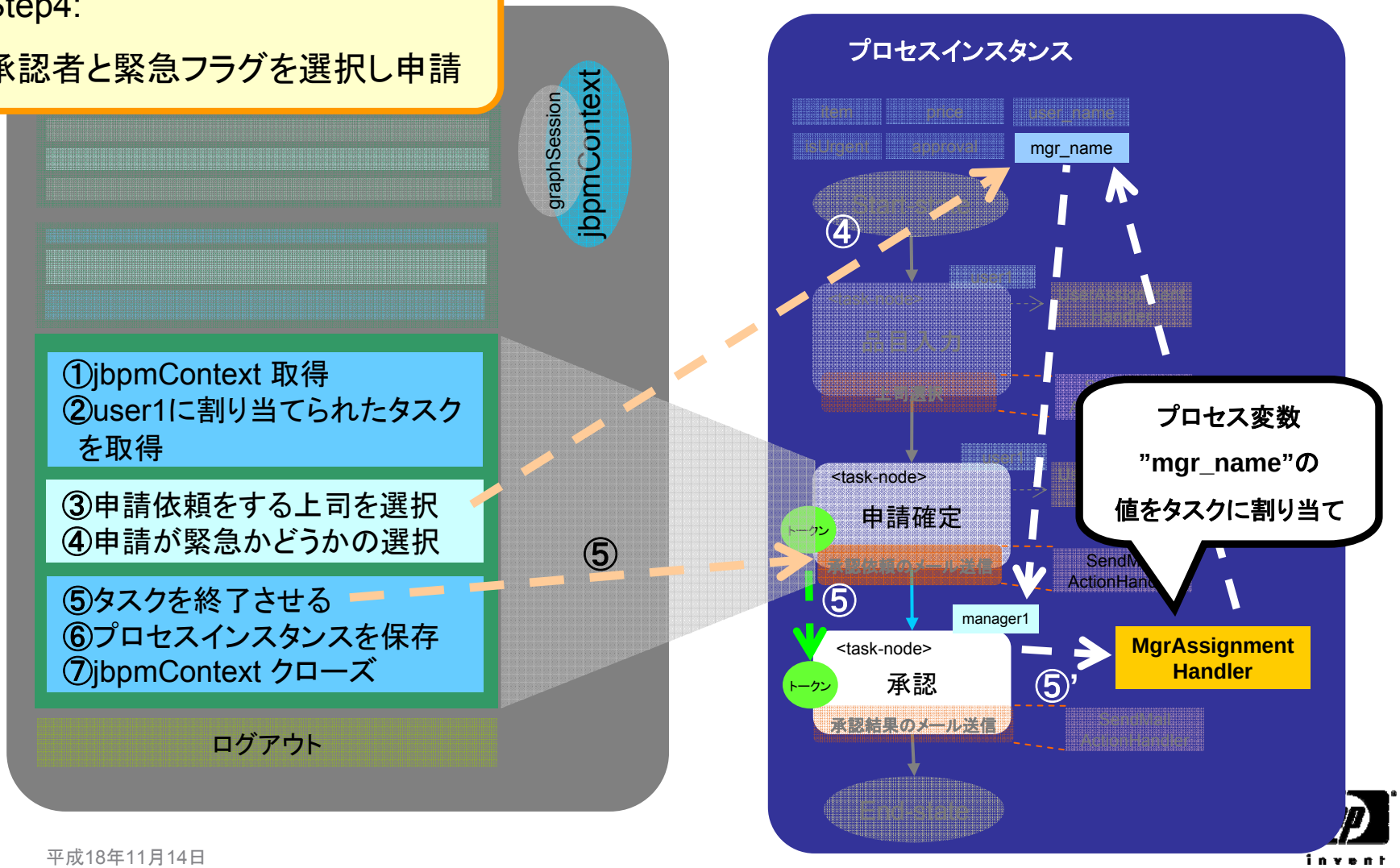
Open Source & Linuxのページ

http://www.hp.com/jp/linux/

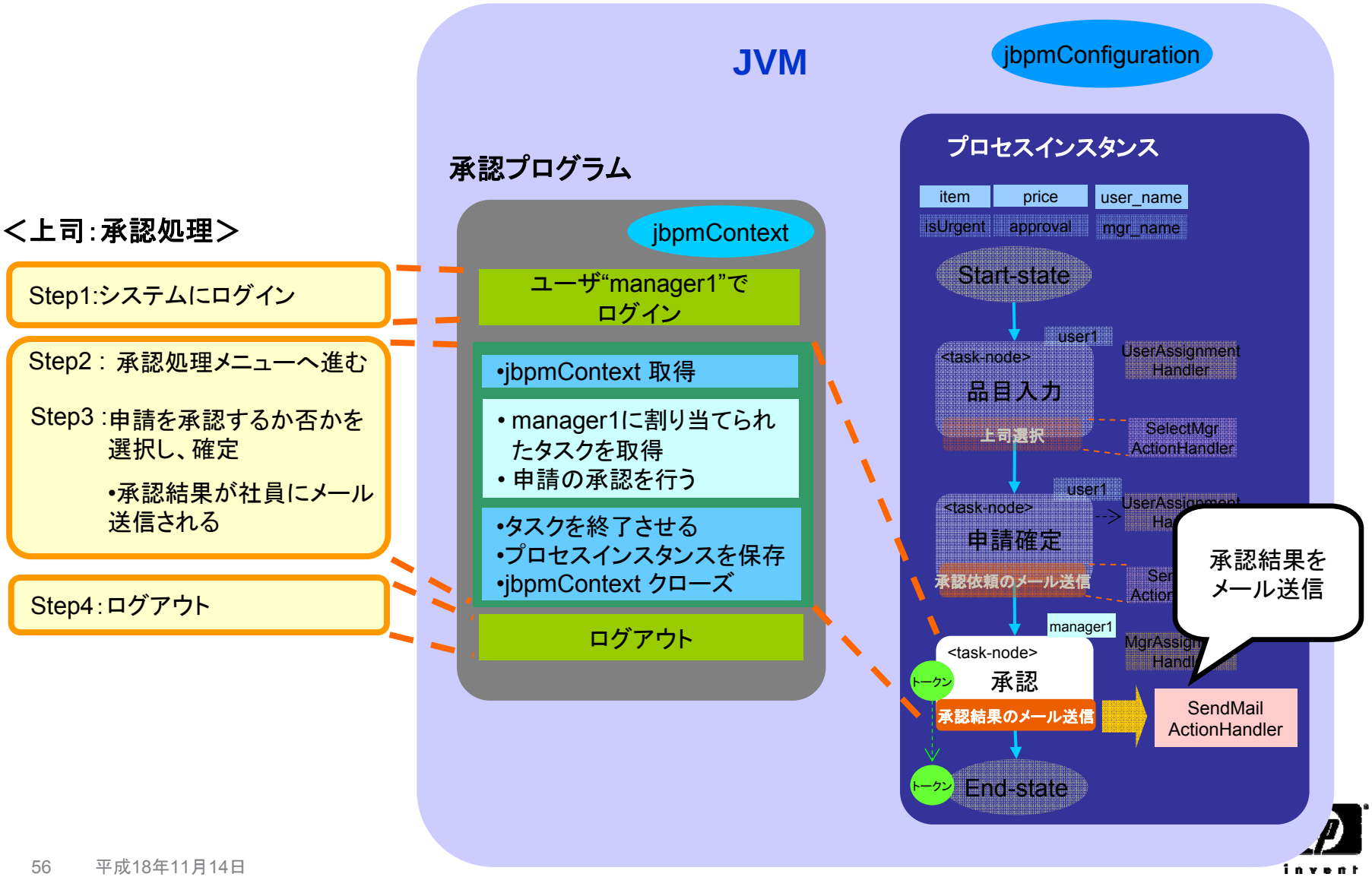
承認タスクへのユーザー登録

Step4:

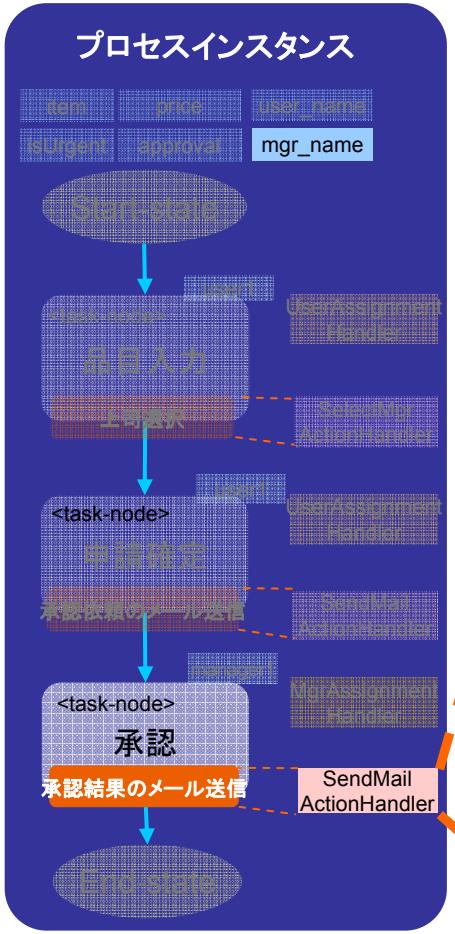
承認者と緊急フラグを選択し申請



承認処理の全体図

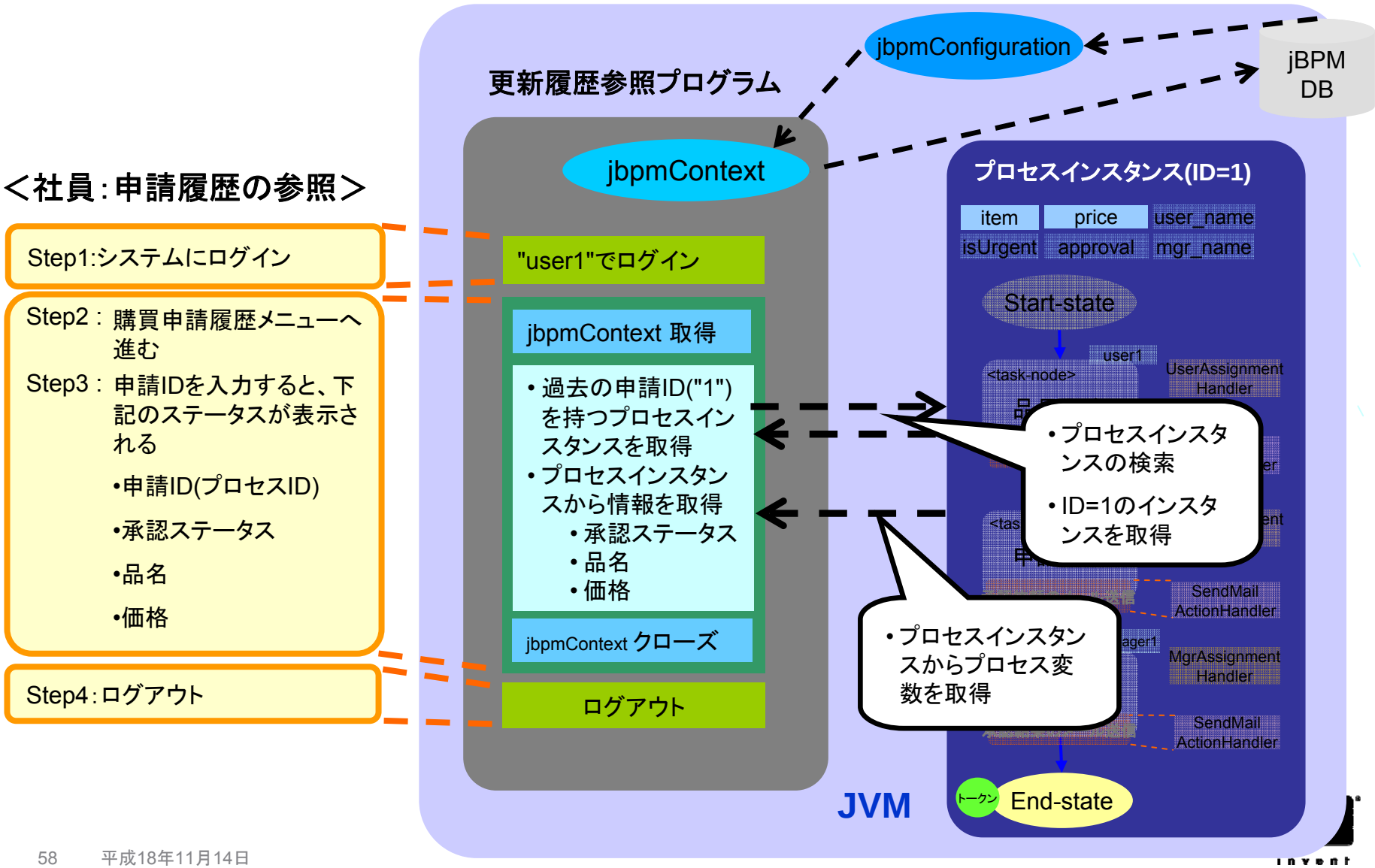

＜上司：承認処理＞
Step1:システムにログイン
Step2： 承認処理メニューへ進む
Step3 :申請を承認するか否かを選択し、確定
•承認結果が社員にメール送信される
Step4：ログアウト
JVM
承認プログラム
jbpmContext
ユーザ“manager1”でログイン
•jbpmContext 取得
• manager1に割り当てられたタスクを取得
• 申請の承認を行う
•タスクを終了させる
•プロセスインスタンスを保存
•jbpmContext クローズ
ログアウト
jbpmConfiguration
プロセスインスタンス
item
price
user_name
isUrgent
approval
mgr_name
Start-state
user1
<task-node>
品目入力
UserAssignment Handler
上司選択
SelectMgr ActionHandler
user1
<task-node>
申請確定
承認依頼のメール送信
manager1
<task-node>
承認
トークン
承認結果のメール送信
SendMail ActionHandler
トークン
End-state
承認結果をメール送信

アクションハンドラーのコード
SendMailActionHandler

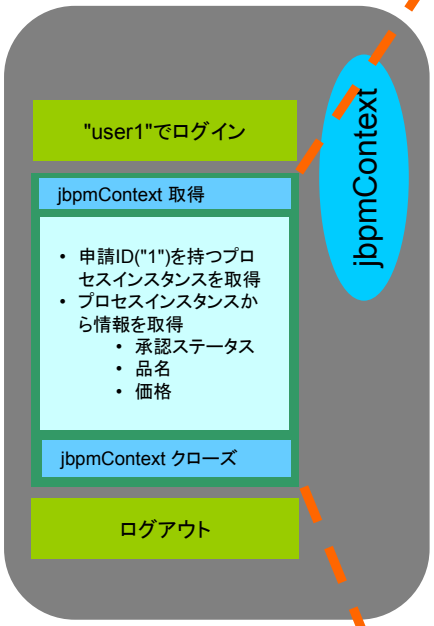



```
import org.jbpm.graph.def.ActionHandler;
import org.jbpm.graph.exe.ExecutionContext;

public class SendMailActionHandler implements ActionHandler {
  public void execute(ExecutionContext executionContext) {

    // 申請者名と承認結果をプロセス変数から取得
    String applicant =
          (String)executionContext.getVariable("user_name");
    String approval =
          (String)executionContext.getVariable("approval");

    // 申請者にメールを送信
    doSendMailToUser(applicant, approval);

  }
}
```

申請履歴を参照するプログラム



申請履歴を参照するプログラム(コード)




```
// jbpmContextの取得
JbpmContext jbpmContext = jbpmConfiguration.createJbpmContext();
GraphSession graphSession = jbpmContext.getGraphSession();
```

```
try {
  String status = "processing";
  String item = "none";
  String price = "none";
  long pid = getPid(); // プロセスインスタンスIDを取得

  // プロセスインスタンスIDからプロセスインスタンスを取得
  ProcessInstance pi = graphSession.loadProcessInstance(pid);
  if (pi != null) {
    // 品目、価格の取得
    item = (String)pi.getContextInstance().getVariable("item");
    price = (String)pi.getContextInstance().getVariable("price");
    // プロセスインスタンスが終了していれば、承認結果を取得
    if (pi.hasEnded()) {
      status =
       (String)pi.getContextInstance().getVariable("approval");
    }
  }
  printStatus(pid, status, item, price);
```

```
} finally {
  jbpmContext.close(); // jbpmContextのクローズ
}
```


invent