ChIP: a Choreographic Integration Process

Saverio Giallorenzo\textsuperscript{1}, Ivan Lanese\textsuperscript{2,3}, and Daniel Russo\textsuperscript{3}

\textsuperscript{1} University of Southern Denmark, Denmark
\textsuperscript{2} Focus Team, INRIA
\textsuperscript{3} University of Bologna, Italy
Context: System Integration
Context: System Integration

Bank Inc.
Context: System Integration

Bank Inc.
Context: System Integration
Context: System Integration
Context: System Integration

Bank Inc.
Context: System Integration

Bank Inc.

API

ChIP: a Choreographic Integration Process
Context: System Integration

Bank Inc.

API

ChIP: a Choreographic Integration Process
Context: System Integration

Bank Inc. [Diagram]

Connector

API

ChIP: a Choreographic Integration Process
Context: System Integration

Bank Inc.

Connector

API
Context: System Integration

Bank Inc.

Card Issuer Inc.

Connector

API
Context: System Integration

Bank Inc.  
Card Issuer Inc.

Connector  
API
Context: System Integration

Bank Inc.

Card Issuer Inc.

Shopping Inc.

Connector

API

ChIP: a Choreographic Integration Process
CoopIS 2018
Context: System Integration

Who coordinates this conversation?
Context: System Integration

Traditional Approach

Global Specification

Local Connector \(1\)

Local Connector \(n\)

Local choices (might be contrasting among collaborating parties)

ChIP Approach

Global Specification

Global Testing

Local Connector \(1\)

Local Connector \(n\)

One-sided observations

Legend:
Research Question

Can we **fill** these **information** gaps **AND** preserve **separation of concerns**?

Local choices (might be contrasting among collaborating parties)

one-sided observations

Discrepancies among disparate technology stacks. The interest in this scenario has recently increased thanks to a new revenue model, called API Economy. Adopting API Economy, many companies, among which Google, Facebook, eBay, and Sabre, started to sell to other organizations the access (under subscription, license, etc.) to their internal services. Although API Economy founds its model on integration, its practice is left to unstructured glue programming. Both centralized and distributed glue programs are used. In the centralized approach a unique glue program interacts with all the integrated systems. This entails all classical drawbacks of centralized systems in terms of scalability and reliability. Furthermore, this centralized connector has full access to the integrated functionalities, yet it resides on the premises of one of the involved organizations (or on third-party premises), hence issues of trust among organizations arise.

In the distributed approach, each stakeholder provides one or more connectors. Each connector interacts with both (i) other connectors, to realize the intended logic of integration and (ii) a set of local functionalities, which may not be accessible by the other connectors. In essence, each connector acts as an adapter for other glue programs. Each connector runs on the premises of one of the collaborating partners, providing a controlled access towards one or more of its resources.

The traditional approach in developing distributed glue programs is represented in Figure 1, top half. First, a global specification of the integration is agreed upon by the stakeholders, using (frequently informal or semi-formal) notations such as Message Sequence Charts (MSC), UML Sequence Diagrams or BPMN choreographies. The development team of each partner uses such a global specification as a reference to build the local implementation of the glue programs. Finally, the network of glue programs is executed and the emerging behavior is contrasted against the global specification.

The main drawback of this approach lies in the huge information gap between a global specification and its distributed implementation. Given a shared global specification, the development team of each partner independently fills local choices (might be contrasting among collaborating parties).

Research Question

**Can we fill these information gaps AND preserve separation of concerns?**
Recent theoretical investigations explored how the interaction- and process-oriented
choreographies, where we use the notations

\[
\begin{align*}
\text{Bank} & \to \text{Card Issuer : } \text{validation}; \\
\text{Card Issuer} & \to \text{Bank : } \text{approval}
\end{align*}
\]

To exemplify the difference between the interaction-oriented and the process-oriented
choreographies we use the notations

\[
\begin{align*}
\text{Bank} & \to \text{Card Issuer : } \text{validation}; \\
\text{from Card Issuer : } \text{approval} \\
\text{Card Issuer} & \to \text{Bank : } \text{validation}; \\
\text{from Bank : } \text{approval}
\end{align*}
\]
Programming

ChIP: a Choreographic Integration Process

Bank → Card Issuer: \textit{validation}; Card Issuer → Bank: \textit{approval}

Bank to Card Issuer: \textit{validation}; from Card Issuer: \textit{approval}

Card Issuer from Bank: \textit{validation}; to Bank: \textit{approval}

Automatic Synthesis of Connectors

Bank Inc. Card Issuer Inc.
ChIP: a Choreographic Integration Process

A choreographic programming language is compatible with ChIP if it supports the integration of external functions (provided by the collaborating parties).
ChIP: a Choreographic Integration Process

Programming

include `verifyRequest` from ...

Bank → Card Issuer: `validation`;
approval = `verifyRequest`@Card Issuer( validation );
Card Issuer → Bank: `approval`

Each local connector knows when and how to interact with its local system.
ChIP: a Choreographic Integration Process

include `verifyRequest` from ...

Bank $\rightarrow$ Card Issuer : `validation`;

approval = `verifyRequest`@Card Issuer( validation );

Card Issuer $\rightarrow$ Bank : `approval`

The Card Issuer will not provide this information to the other parties, since it is necessary only when it (locally) generates its local connector.
Abstract.

Over the years, organizations acquired disparate software systems, each answering one specific need. Currently, the desirable outcomes of integrating these systems (higher degrees of automation and better system consistency) are often outbalanced by the complexity of mitigating their discrepancies. These problems are magnified in the decentralized setting (e.g., cross-organizational cases) where the integration is usually dealt with ad-hoc "glue" connectors, each integrating two or more systems. Since the overall logic of the integration is spread among many glue connectors, these solutions are difficult to program correctly (making them prone to misbehaviors and system blocks), maintain, and evolve.

In response to these problems, we propose ChIP, an integration process advocating choreographic programs as intermediate artifacts to refine high-level global specifications (e.g., UML Sequence Diagrams), defined by the domain experts of each partner, into concrete, distributed implementations. In ChIP, once the stakeholders agree upon a choreographic integration design, they can automatically generate the respective local connectors, which are guaranteed to faithfully implement the described distributed logic. In the paper, we illustrate ChIP with a pilot from the EU EIT Digital project SMAll, aimed at integrating pre-existing systems from government, university, and transport industry.

1 Introduction

Over the years organizations acquired several software systems, each satisfying one specific need. Traditionally these systems hardly integrate with each other due to incompatible technology stacks [20, 23]. It has been empirically observed that this leads to system stratification and increasing technical debt [42]. Contrarily, the high level of automation and consistency achievable by the integration of such systems could satisfy new requirements, maximize business/service performance, and avoid duplication of resources. This is confirmed by the thriving economics of Enterprise Resource Planners (ERPs) [26]. ERPs offer a closed, rigid yet highly structured environment for system integration. However, ERPs are rarely a solution for cross-organizational integration, where the enforcement of a unique platform is nearly impossible.

In cross-organizational settings, the only possible approach is given by mediating applications, usually called "glue" programs [18] or connectors, that mitigate programming local choices regard only the deployment information to let connectors interact with local systems. Connectors execute the exact behaviour defined in the choreography.

Local choices regard only the deployment information to let connectors interact with local systems.

Expected Global Behavior
An Illustrative Example

**Regional Government**
- Tracked line
- **loop** (until schedule has next stop)
- Calculated delay

**Tracker by University**
- Line schedule
- Current bus position

**Bus Agency**

Global Specification

Choreographic Program (Global Design)

Choreographic Program + Local Bindings
- local refinement

Local Connector
- selective compilation

Expected Global Behavior

ChIP Approach

Traditional Approach

Legend:
- Manual step
- Automatic step

Project description: https://forumvirium.infini.fox/small4/develops/mobility/services

Deployable platform: https://hub.docker.com/u/smallproject

Documentation: https://github.com/small/developers/wiki
An Illustrative Example

```
1 setLine: Government(line) -> BusAgency(line);
2 passSchedule: BusAgency(shd) -> Tracker(shd);
3 while(hasNext)@
  4 passPosition: BusAgency(pos) -> Tracker(pos);
  5 storeDelay: Tracker(delay) -> Government(delay)
```

Choreographic Program + Local Bindings

Global Specification

Execution

Expected Global Behavior
An Illustrative Example

locations {
    Admin: "reg-gov.org:80/BusCheckAdmin"
    Tracker: "university.edu:80/Tracker"
    BusAgency: "bus-agency.com:80/BusCheck"
}

// code

shd@BusAgency = getSchedule(line);
passSchedule: BusAgency(shd) -> Tracker(shd);
hasNext@Tracker = hasNextStop(shd);
while(hasNext@Tracker) {
    pos@BusAgency = getPosition(line);
    passPosition: BusAgency(pos) -> Tracker(pos);

An Illustrative Example

deployment {
  getSchedule from "socket://intranet.schdls:8000" with SOAP
  getPosition from "socket://intranet.GPS:8001" with HTTP
}

locations {
  Admin: "reg-gov.org:80/BusCheckAdmin"
}

// code

shd@BusAgency = getSchedule(line);
passSchedule: BusAgency(shd) -> Tracker(shd);
hasNext@Tracker = hasNextStop(shd);
while(hasNext)@Tracker {
  pos@BusAgency = getPosition(line);
  passPosition: BusAgency(pos) -> Tracker(pos);
}
An Illustrative Example

deployment {
  getSchedule from "socket://intranet.schdl:8000" with SOAP
  getPosition from "socket://intranet.GPS:8001" with HTTP 
}

locations {
  Admin: "reg-gov.org:80/BusCheckAdmin"
  DatabaseConnector: "reg-gov.org:80/BusCheckDB"
}

shd@BusAgency = getSchedule(line);
passSchedule: BusAgency(shd) -> Tracker(shd);
hasNext@Tracker = hasNextStop(shd);
while(hasNext)@Tracker {
  pos@BusAgency = getPosition(line);
  passPosition: BusAgency(pos) -> Tracker(pos);
}

Compilation to an executable program

Global Specification

Choreographic Program (Global Design)

Choreographic Program + Local Bindings

Local Connector

Local Connector

Choreographic Program + Local Bindings

execution

Expected Global Behavior
Evolving ChIP systems

It’s a matter of re-iterating over (some) steps of the process

Regional Government  Tracker by University  Bus Agency

Tracked line

Line schedule

loop (until schedule has next stop)
alternative (if signed data is valid)

Calculated delay

Signed current bus position

Choreographic Program + Local Bindings_1

Local Connector_1

Choreographic Program + Local Bindings_n

Local Connector_n

Global Specification
refinement

Choreographic Program (Global Design)

expected behavior
execution

refinement

selective compilation

local grounding
Evolving ChIP systems

It's a matter of re-iterating over (some) steps of the process

```java
while (hasNext) @Tracker {
    pos @BusAgency = getPosition(line);
    signed_pos @BusAgency = sign(pos);
    passPosition: BusAgency(signed_pos) -> Tracker(pos);
    valid @Tracker = validate(pos);
    if (valid) @Tracker{
        delay @Tracker = calculateDelay(shd, pos);
        storeDelay: Tracker(delay) -> DatabaseConnector(delay);
    } @DatabaseConnector = insertDelay(line, delay)
};
hasNext @Tracker = hasNextStop(shd)
```

Dynamic Evolution.

As exemplified above (as well as in the traditional approach), when updating a pre-existing system, the stakeholders have to shut-down the deployed connectors, replace them, and restart the whole integrated architecture. In many application contexts these downtimes are acceptable, however when the integrated systems need to be always online, it is imperative to avoid or at least minimize these downtimes.

Shutdown can be avoided by resorting to live update techniques, such as the ones described in [44]. However, as shown in [44], only a few of these techniques support the live update of distributed systems. Indeed, while the centralized case entails the update of just one program, in the distributed case a protocol must be in place to coordinate the update of the distributed components, avoiding

Legend:
Evolving ChIP systems

It’s a matter of re-iterating over (some) steps of the process

while(hasNext)@Tracker { 
    pos@BusAgency = getPosition(line);
    signed_pos@BusAgency = sign(pos);
    passPosition: BusAgency(signed_pos) -> Tracker(pos);
    valid@Tracker = validate(pos);
    if( valid )@Tracker{
        delay@Tracker = calculateDelay(shd, pos);
        storeDelay: Tracker(delay) -> DatabaseConnector(delay);
        _@DatabaseConnector = insertDelay(line, delay)
    }
    hasNext@Tracker = hasNextStop(shd)
}
Evolving ChIP systems

It’s a matter of re-iterating over (some) steps of the process

```
while(hasNext)@Tracker {
    pos@BusAgency = getPosition(line);
    signed_pos@BusAgency = sign(pos);
    passPosition: BusAgency(signed_pos) -> Tracker(pos);
    valid@Tracker = validate(pos);
    if( valid )@Tracker{
        delay@Tracker = calculateDelay(shd, pos);
        storeDelay: Tracker(delay) -> DatabaseConnector(delay);
        _@DatabaseConnector = insertDelay(line, delay);
    }
    hasNext@Tracker = hasNextStop(shd)
}
```

Only the BusAgency and the Tracker will update their local groundings and re-compile their connectors
Evolving ChIP systems

It’s a matter of re-iterating over (some) steps of the process

```plaintext
while (hasNext) @ Tracker {
    pos @ BusAgency = getPosition(line);
    signed_pos @ BusAgency = sign(pos);
    passPosition: BusAgency(signed_pos) -> Tracker(pos);
    valid @ Tracker = validate(pos);
    if ( valid ) @ Tracker{
        delay @ Tracker = calculateDelay(shd, pos);
        storeDelay: Tracker(delay) -> DatabaseConnector(delay);
        _ @ DatabaseConnector = insertDelay(line, delay)
    }
    hasNext @ Tracker = hasNextStop(shd)
}
```

Only the BusAgency and the Tracker will update their local groundings and re-compile their connectors
Conclusion and Future Work

- Which specification languages are better suited* for the ChIP refinement process?
- Can we provide guidelines for the refinement?
- Can we automatise (parts of) the refinement?

* ease of transition, expressiveness, automatisation, …

- Enhance AIOCJ with richer data structures and type system
- Capture other aspects of the design:
  - Security
  - Exceptional behaviours
  - Non-functional properties
  - …

Legend: