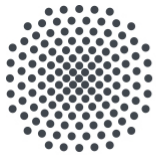
Pattern-based Rewrite and Refinement of Architectures Using Graph Theory

IAAS Research

Summer SoC 2019, Crete, Greece



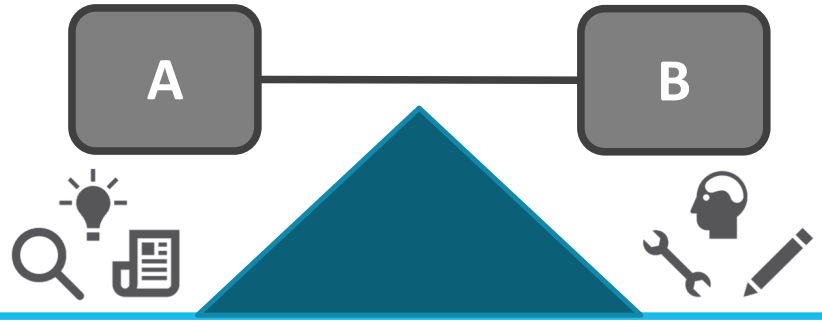
University of Stuttgart

Jasmin Guth and Frank Leymann

{guth, leymann}@iaas.uni-stuttgart.de

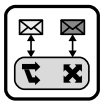
Institute of Architecture of Application Systems

Introduction & Motivation



3.6.2 Message-oriented Middleware

Asynchronous message-based communication is provided while hiding complexity resulting from addressing, routing, or data formats from communication partners to make interaction robust and flexible.



How can communication partners exchange information asynchronously with a communication partner?

Context

The application components of a *distributed application* (139) are hosted on multiple cloud resources and have to exchange information with each other. Often, the integration with other cloud applications and non-cloud applications is also required. These different applications possibly use different programming languages, data formats, and *execution environments* (94). When one application directly exchanges information with another application, the address and data format of the target application has to be respected. Even within one homogeneous *distributed application* (139), the communicating application components must be available at the time when the information shall be exchanged. These dependencies can significantly reduce the availability of the overall application as the failure of one application component would directly affect all components communications with it. The resulting dependency between communication partners regarding their location, availability, and data format is called *tight coupling*. It also increases the complexity of the management of the overall application or the landscape of applications, because changes to one communication partner, for example, regarding the format of exchanged data or the address used, also affect the other communication partner directly. This should be avoided to increase the availability of the overall application and ease continuous alterations by making communication flexible.

Solution

Communication partners exchange information asynchronously using messages handled by a *message-oriented middleware*. For this purpose, a *message-oriented*

A pub-sub channel [1] may be used to broadcast a message to multiple receivers. While a message queue conceptually delivers messages to only one receiver, a pub-sub channel delivers messages to multiple receivers.

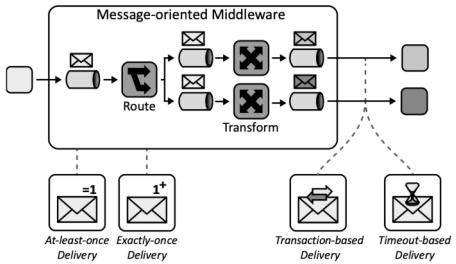


Figure 37: Message-oriented Middleware and Related Patterns

Result

When interacting with a *message-oriented middleware*, a sender puts a message on one message queue or pub-sub channel and receivers can retrieve it from possibly different queues. In between these two access points the *message-oriented middleware* handles the complexity of addressing, availability of communication partners and message format transformation as shown in Figure 37. Therefore, in addition to message queues, the *message-oriented middleware* provides components that route messages to intended receivers as well as handle message format transformation. Communication partners may communicate via messages without the need to know the message format expected by the communication partner or the address at which it can be reached. Furthermore, communication partners can send and receive messages at their own pace and without relying on the availability of communication partners.

[1]The *message-oriented middleware*, therefore, suggests a *pipes-and-filters* application architecture as covered by Hohpe and Woolf [1], Bushmann et al. [14] and in the *distributed application* (143) pattern in Chapter 4. In this scope, application components act as independently operating *filters* that are interconnected through *pipes*, i.e., the message queues provided by the *message-oriented middleware*. Hohpe and Woolf [1] describe the behavior of these pipes and how they may be connected. Regarding the *filters*, i.e., the application components, Hohpe and Woolf also described how to interface with the messaging systems in the *adapter* pattern.

The more intermediaries a message passes through while traversing a *message-oriented middleware*, the more likely it becomes that an intermediary fails. To address this issue, messages are often stored in persistent storage by *the message-oriented middleware* from where they can be recovered in case of failures. This approach is described by the *guaranteed delivery* pattern introduced by Hohpe and Woolf [1].

Variations

A *message-oriented middleware* is used if small amounts of data need to be exchanged frequently, as messages are often restricted in size so they can be handled more easily. If larger amounts of data have to be exchanged, messages may either contain a pointer to this data that is actually stored at a different location, for example, a storage offering (see Section 3.5) or the data may be split up among multiple messages. Hohpe and Wolf [1] cover patterns for this exchange of large data elements: the *file transfer* pattern describes how data may be exported from one application and imported by a different one. A *message sequence* may be used to split large data elements among a set of messages.

Related Patterns

Figure 37 shows how the other messaging patterns of this section are related to the message oriented middleware:

- *At-least-once delivery* (128): it is ensured that messages traversing the *message-oriented middleware* are delivered once or multiple times. This is achieved through acknowledgements for message receives. If an acknowledgement is not received, a message is retransmitted.
- *Exactly-once delivery* (126): messages traversing the message-oriented middleware are delivered once and only once to the receiver. This involves reliably storage of messages in the *message-oriented middleware* and, often, transactional message exchange during its traversal of *the message-oriented middleware*.

In addition to these communication pattern
should be considered to be implemented i
teracting with a *message-oriented middlew*

- *Transaction-based processor* (179): if
transactions to assure that messages
transaction can be extended to include
the receiver as well. Therefore, the *t*
application to assure that messages ar
havior is also described by Hohpe an
tern. The transaction-based processor
extends it to the transactional interacti
- *Timeout-based message processor* (18
assures *at-least-once delivery* (128) b
client can extend the acknowledgmen
Therefore, the *timeout-based message*
sure that messages are processed *at le*
- *Distributed application* (139): applic
loosely coupled (139) application co
oriented middleware to exchange inf
scope, an *idempotent processor* (176
messages created by a *message-orie*
delivery.
- *Message mover component* (201): thi
different *message-oriented middlewa*
providers or that are installed in on-pr
- *Watchdog* (230): a watchdog may be
pecially, in scope of *environment-ba*
queues to store information securely e
- *Batch processing component* (165): m
delay messages. A *batch processing c*
only when conditions are feasible, fo
low or the overall application experien

Known Uses

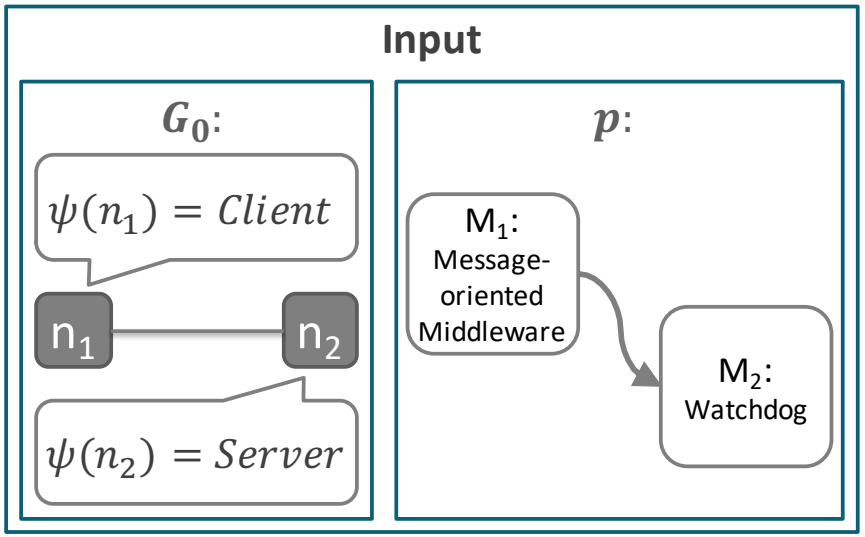
Using messaging to integrate distributed
approach. Many additional messaging patt

C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications.* Springer, 2014.

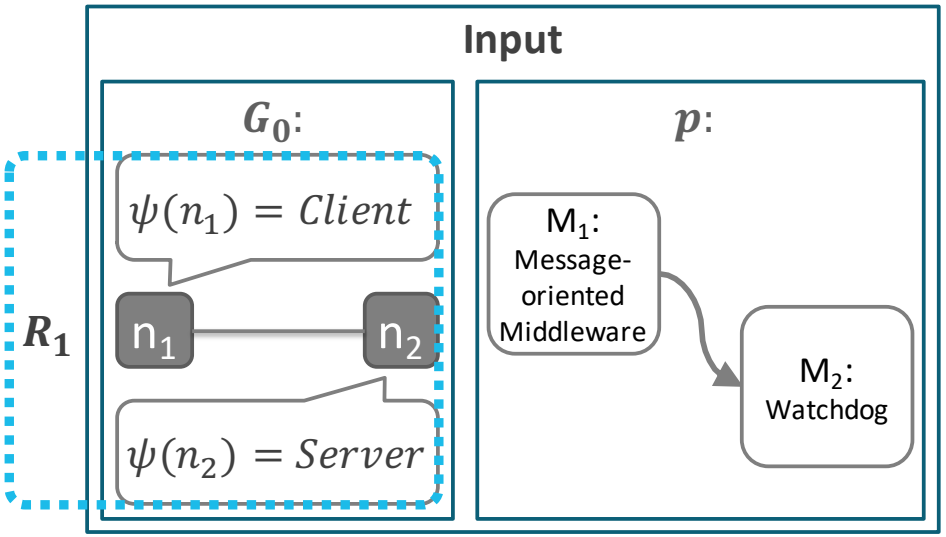
Pattern-based Rewrite and Refinement of Architectures Using Graph Theory

Application Example

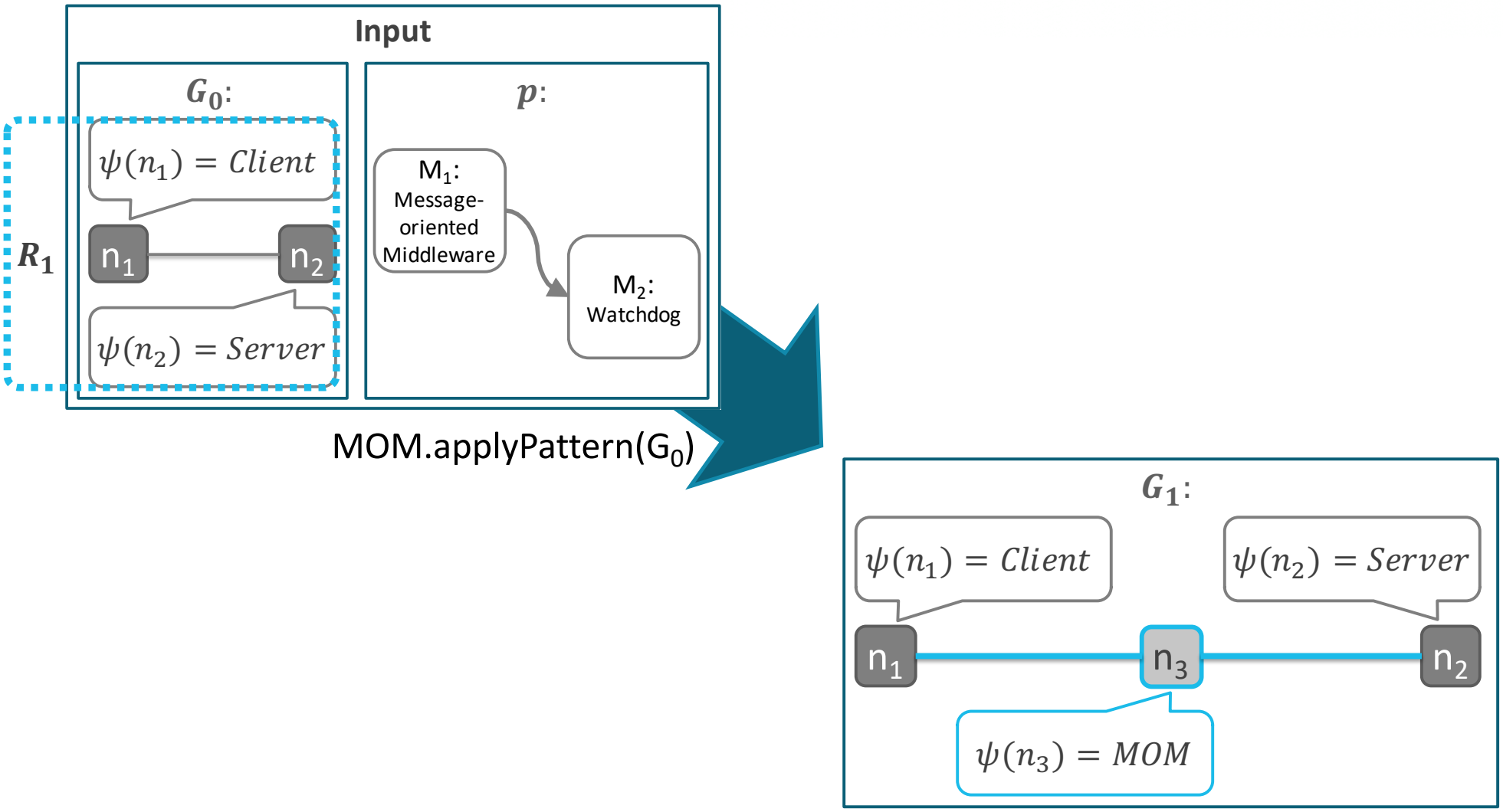
Application Example – *Input*



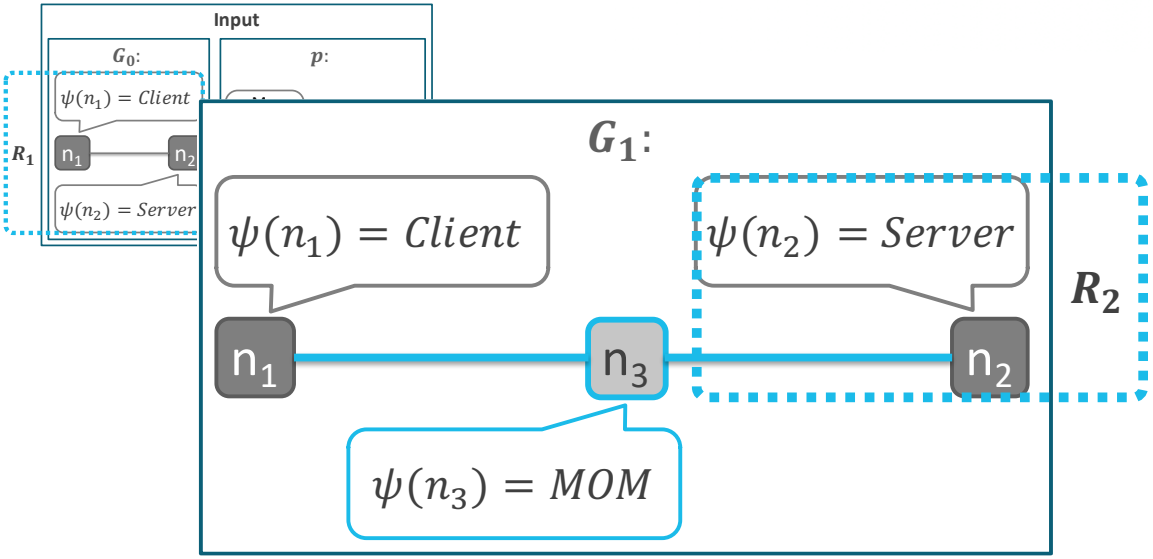
Application Example – M_1: MOM – Check Requirements



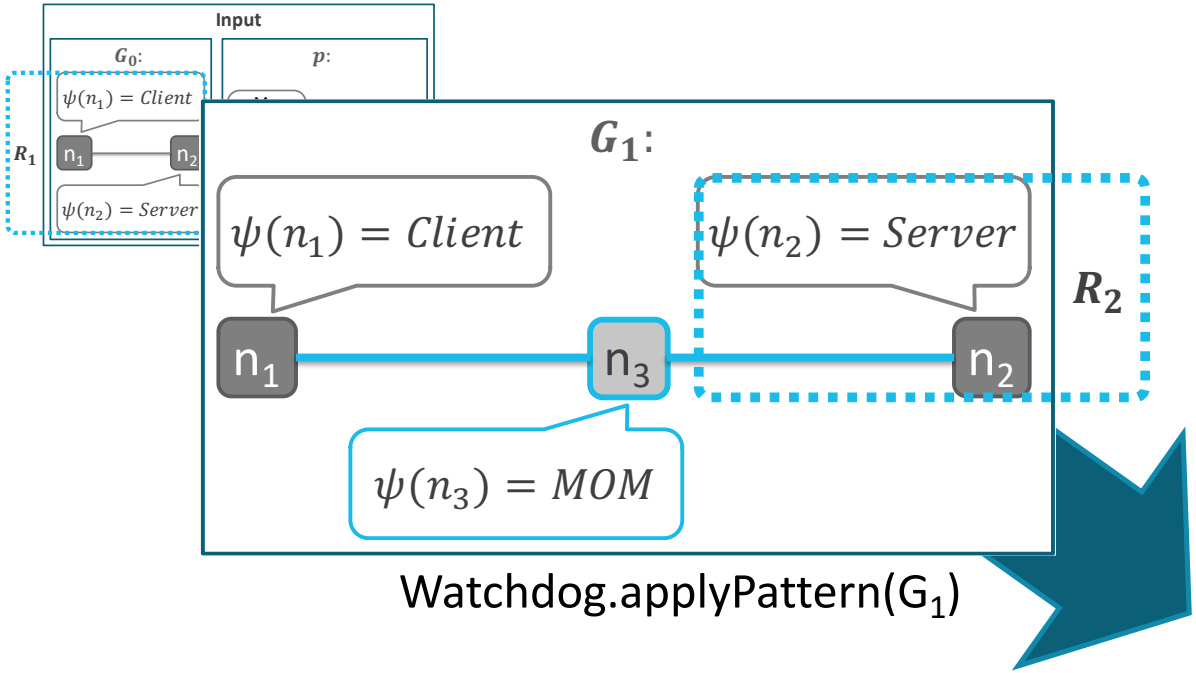
Application Example – M_1: MOM – Refine or Rewrite



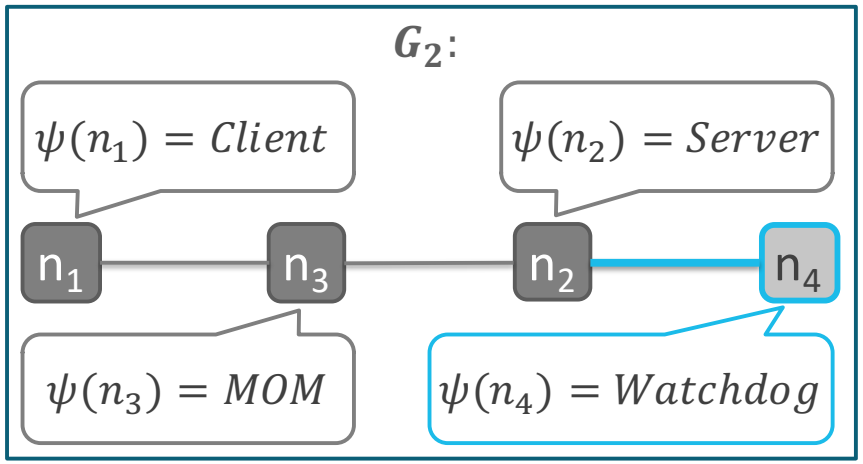
Application Example – M_2: *Watchdog – Check Requirements*



Application Example – M_2: *Watchdog – Refine or Rewrite*



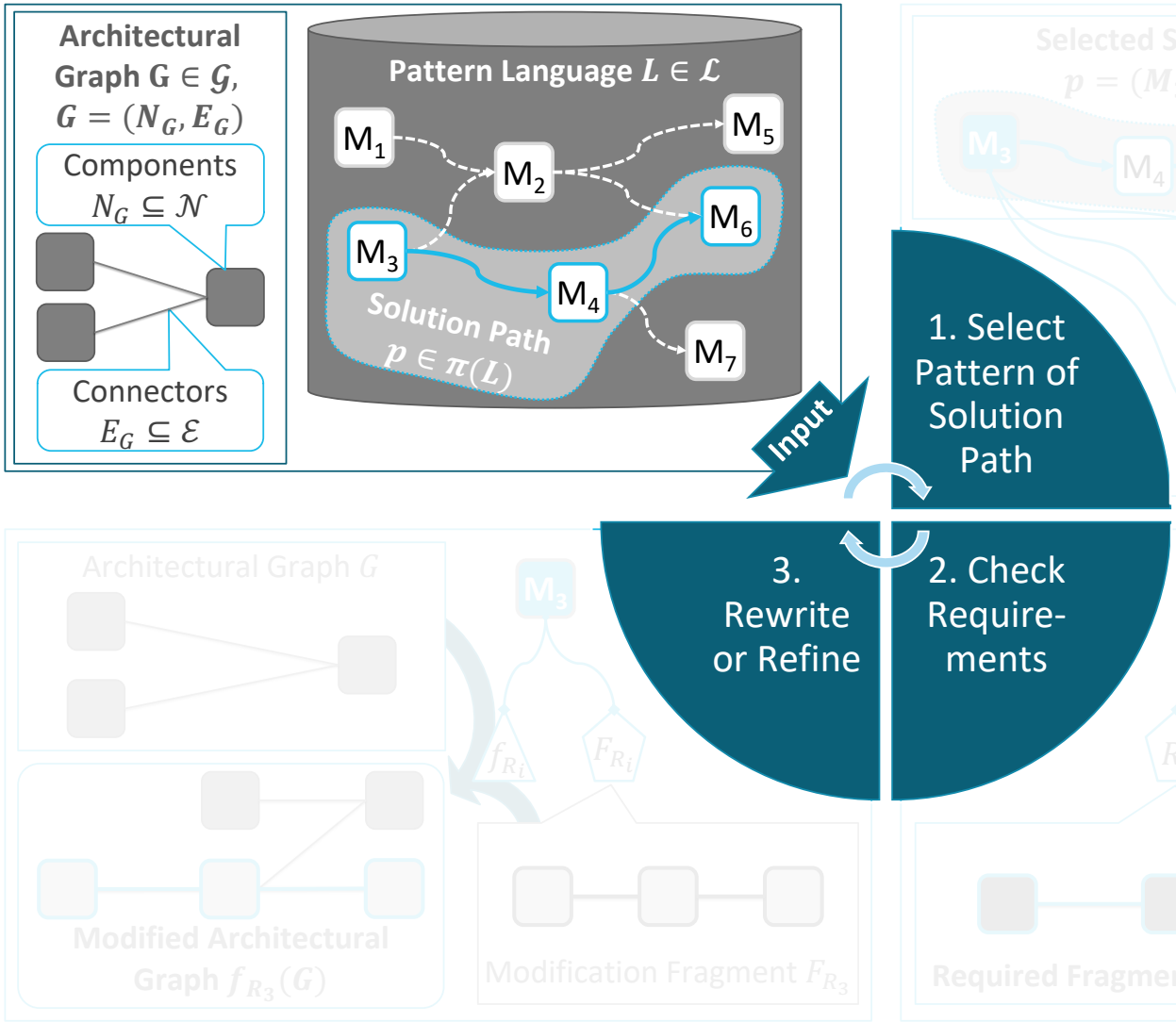
Watchdog.applyPattern(G_1)



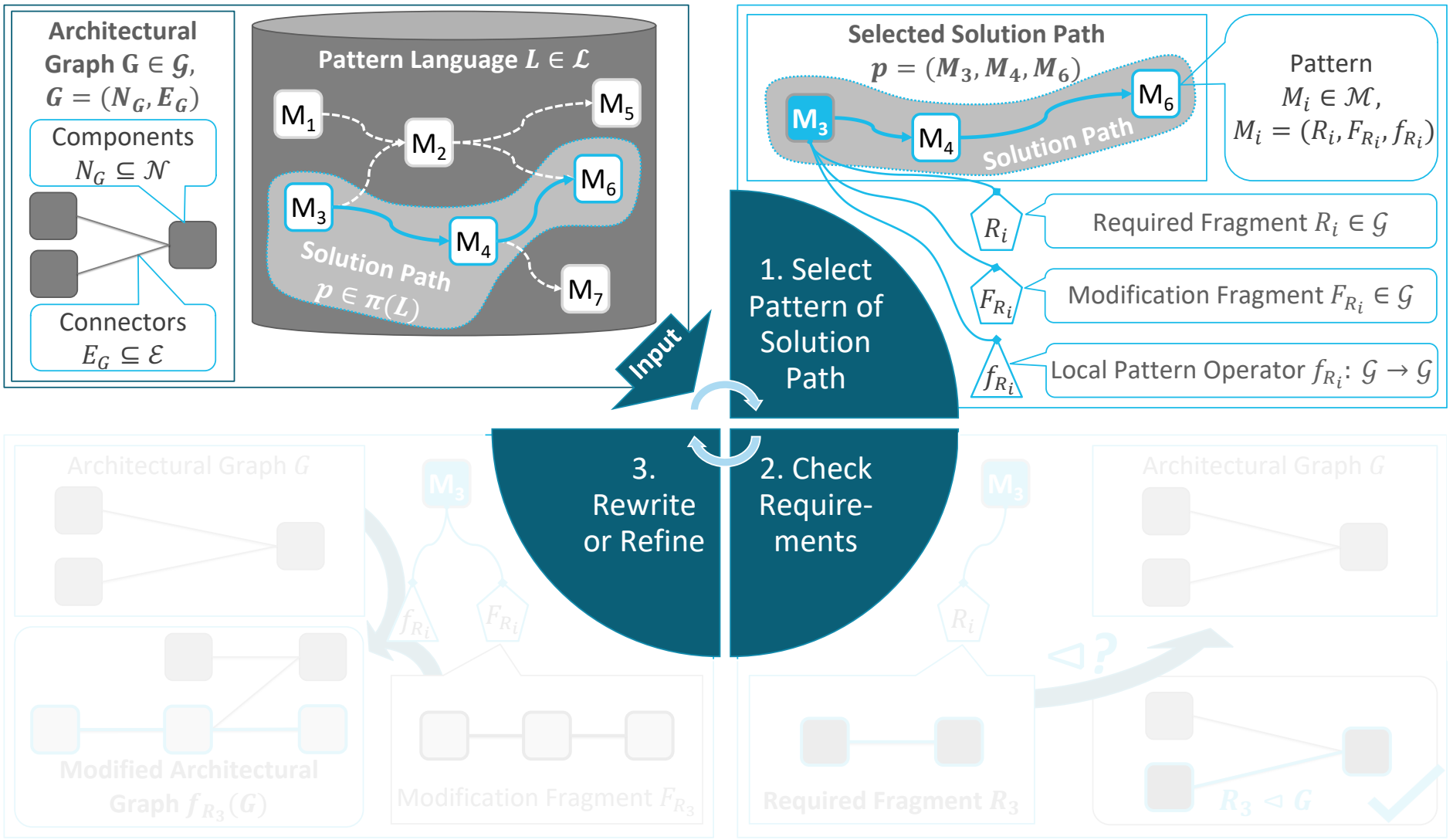
Pattern-based Rewrite and Refinement of Architectures Using Graph Theory

Concept & Formalization

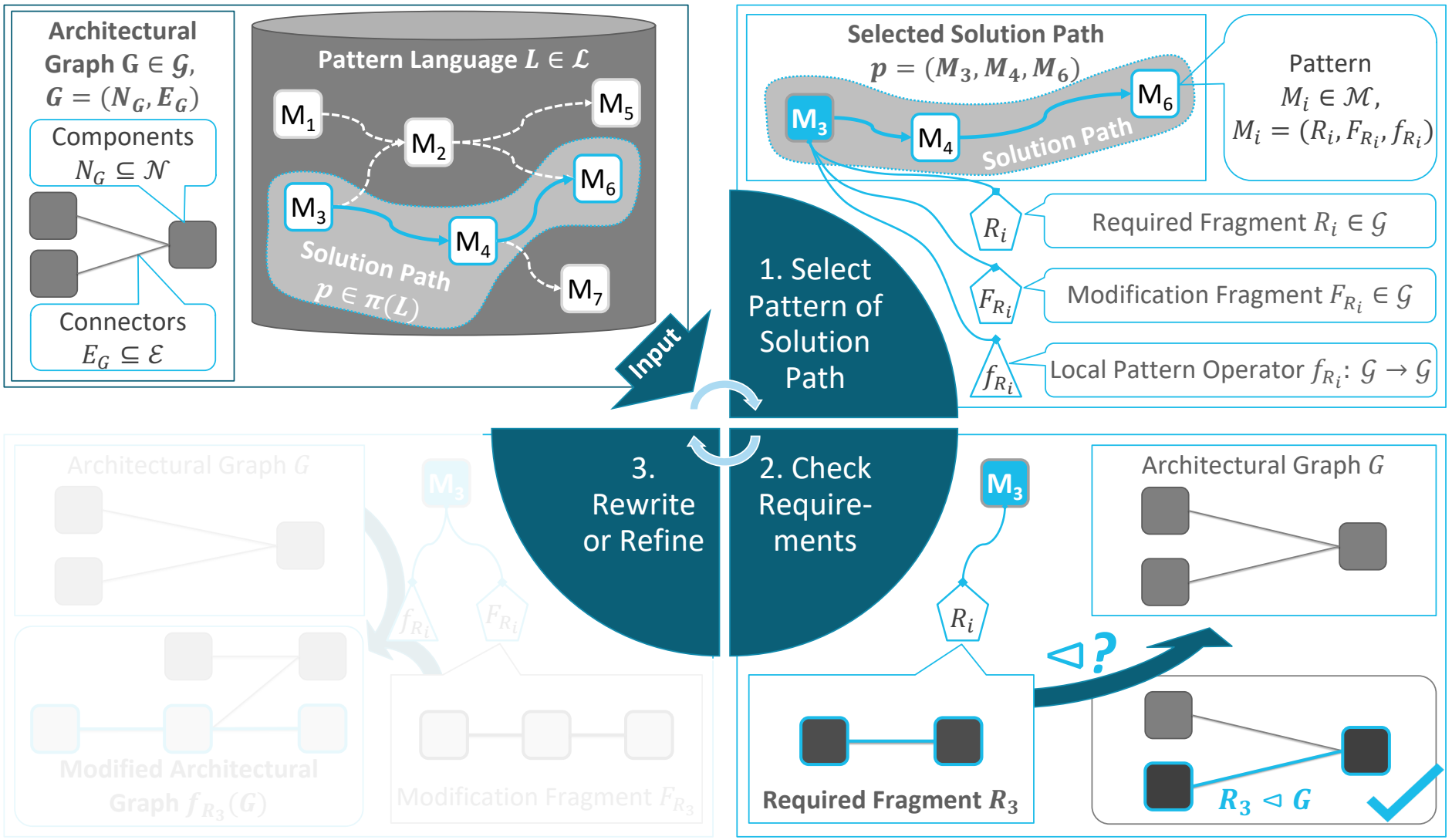
Concept: Pattern-based Rewrite and Refinement of Architectures



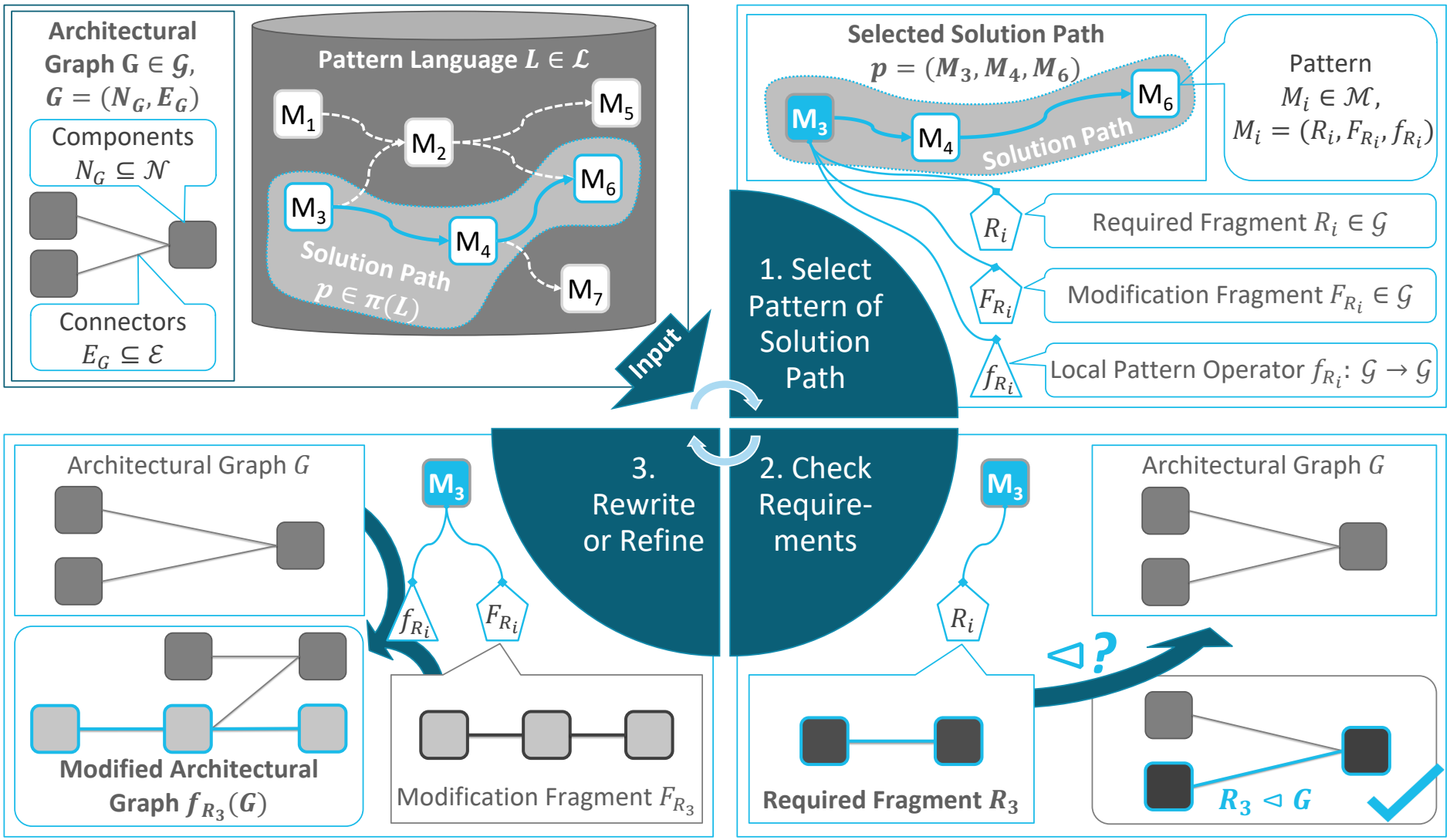
Concept: Pattern-based Rewrite and Refinement of Architectures



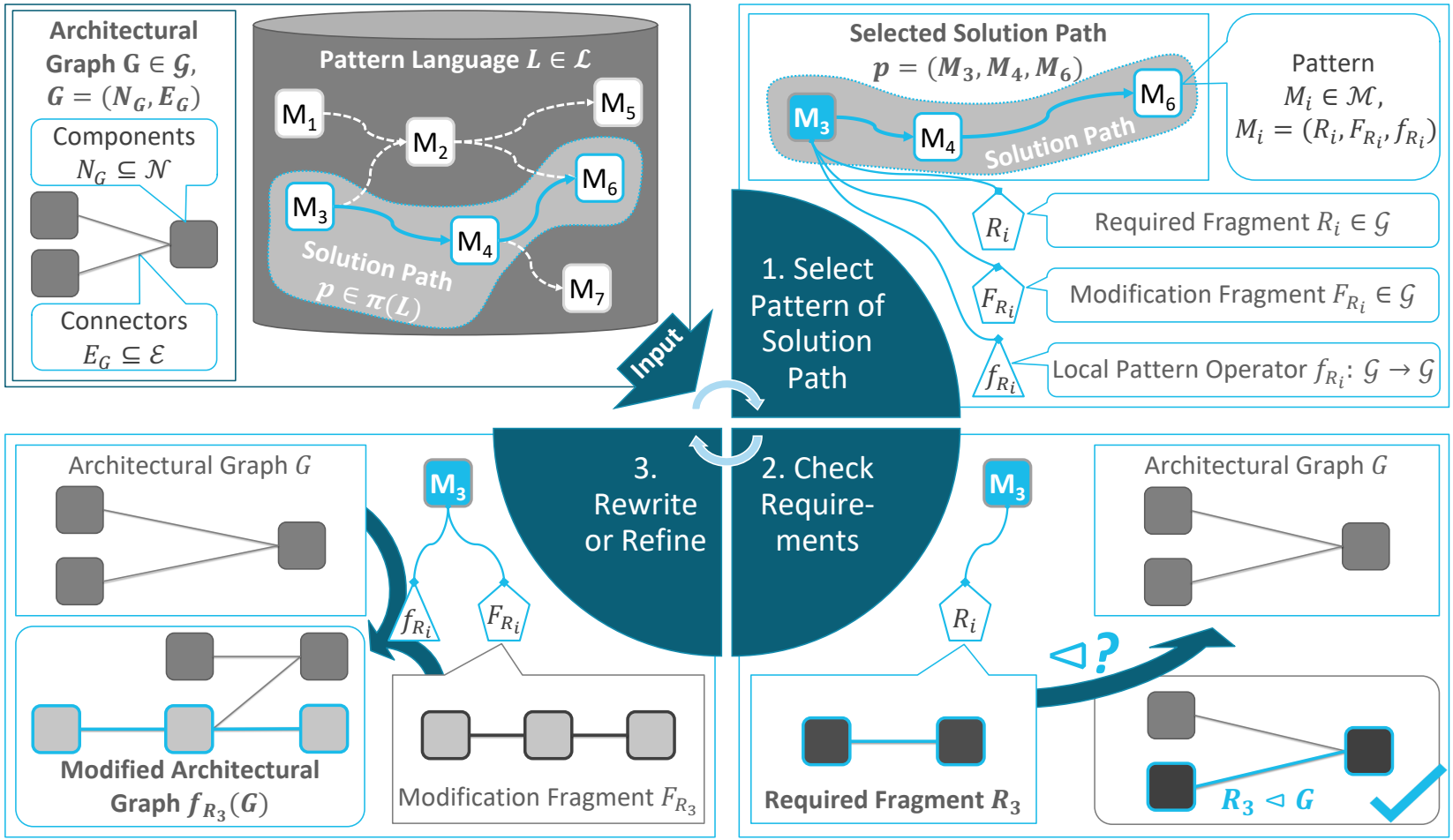
Concept: Pattern-based Rewrite and Refinement of Architectures



Concept: Pattern-based Rewrite and Refinement of Architectures



Conclusion & Future Work – *Poster Session*



Thank you for your attention – I hope for interesting discussions during the poster session!

☺