Testing and Visualizing a Message Queuing Infrastructure

Karl Flieder
Graduate of CAMPUS 02 - University of Applied Sciences, Graz
A-8010 Graz, Sandgassee 23a/28, Austria
karl.flieder@gmx.at

ABSTRACT
Integration on process level has emerged as a promising paradigm for managing Enterprise Application Integration (EAI) strategies. However, there is a missing link between established messaging infrastructures, utilizing the transport of messages and a process-coupled view on it. Testing and analyzing the results are two tightly coupled and normally inseparable mechanisms.

This paper describes an end-to-end test of a messaging system for several business processes achieved through a logical orchestration of the components involved. The messaging model used, including several message types, as well as basic Gestalt principles of perceptual grouping, will be introduced. For visualizing the test results, we present an approach that incorporates principles derived from Gestalt theory. Just as important, enterprise integration patterns and testing issues such as the test model will be explained, too.

The idea of splitting a messaging infrastructure into different layers (business, logical, physical), in order to gain test results for the different business processes, is a novel approach. Likewise, injecting test messages on application level during productive operation is a challenge. Visualizing the test results using Gestalt principles, aids in reducing the cognitive effort. To sum it up, this paper discusses messaging, testing, patterns as well as semantic principles, aids in reducing the cognitive effort. To sum it up, this paper discusses messaging, testing, patterns as well as semantic issues and shows how to visualize a messaging infrastructure for different business processes in a coherent way.

Keywords: Messaging, Testing, Enterprise Integration Patterns, Visualization, Gestalt Theory, Perceptual Grouping.

1. INTRODUCTION
In many cases, messaging – often referred to as Message-Oriented Middleware (MOM) [3], [4] – is used as a base service for integrating distributed systems on application level. For usability reasons, some departments often request their own view of the components involved during operation. System administrators are not always available to answer questions concerning efficiency proof, system availability and further more. Obviously, a customized view of the related messaging components could be advantageous to the department’s super-user. In this paper the author proposes a concept to meet this aim very efficiently without using a professional and expensive system-monitoring tool. The proposed solution focuses on the logical orchestration of the physical message paths involved (Figure 4). It incorporates testing on application level, looking at the results on process level and using Gestalt theory to reduce the cognitive effort for interpreting the test results. Equally important, enterprise integration patterns were considered during design and implementation.

Messaging technology enables asynchronous, high-speed, program-to-program communication with reliable delivery. The two basic components of messaging are messages and queues (Figure 2). Programs communicate by exchanging messages with each other. A sender or producer is a program that sends a message by writing the message into a queue. A receiver or consumer is a program that receives a message by reading it from a queue. This asynchronous communication model makes delivery more reliable and decouples the sender from the receiver. Basically, a message consists of two parts: a header and a body. The header contains metadata about the message; the body contains the payload generated by the sender application.

There are two messaging models available: point-to-point (one-to-one) and publish/subscribe (one-to-many) [7]. Point-to-point messaging is used to send data to a single application. This does not guarantee that every piece of data sent will necessarily go to the same receiver, because the channel might have several receivers. Publish/subscribe provides all the receiver applications with data. In this case, the data is copied for all the receivers. In case of an event, the sender publishes a message; the receivers may subscribe certain topics. Messaging systems are peer-to-peer facilities. Generally, each client can send messages to, and receive messages from any client. Each client connects to a messaging agent that provides facilities for creating, sending and receiving messages. Messaging capabilities are typically provided by a separate software system. This software defines a set of services to mediate between various applications across different platforms. Several systems compete on the market, for example:

- IBM’s WebSphere MQ, can be regarded as the leader on commercial markets [25].
- Java Messaging Service (JMS) as part of the J2EE JDK [16].
- Microsoft’s Message Queuing (MSMQ) as part of the System.Messaging libraries in Microsoft .NET [18].
- Sonic Software’s SonicMQ [21].
- Emerging Web services toolkits that support asynchronous Web services.

JMS addresses the integration of MOM to support messaging in Java object systems. The JMS Application Program Interface (API) comprises a set of interfaces for point-to-point and publish/subscribe messaging. It uses the facilities of underlying messaging systems in the same sense as JDBC is a standard API for accessing relational databases.

EAI is concerned with information integration issues. The intention is to achieve effective communication across various business applications and delivery channels, ensuring seamless data integration from disparate systems. We distinguish between three main levels of integration:
(i) The exchange of data on *data level* relies on data management. Typically, JDBC and ODBC middleware or system interfaces are involved. This type of integration supports data exchange between various data stores when applications in different businesses have to share information. This level of integration can be regarded as the lowest one in enterprise application integration.

(ii) The approach on *application level* operates under the paradigm that applications can easily be wrapped using some form of middleware technology. Integrated applications are built upon this middleware layer where messaging resides. As a matter of fact, this is a powerful concept in software engineering, because middleware represents a layer of abstraction to prevent top layers knowing details about the layers below.

(iii) Integration on *process level* represents the highest level of abstraction. Several applications work together to fulfill a certain business process. This level can be reached in assistance with an integration server dealing with application server tasks. A lot of transformations, mappings and flow-control activities are done this way. Typically, automated processes involve systems and people.

### 2. MESSAGING INTERACTION SCHEME

#### Point-to-Point Mode

In message queuing systems, messages are preferred stored in first-in-first-out (FIFO) queues. Producers append (push) messages asynchronously at the end of a queue, while consumers pull them synchronously at the front of a queue (Figure 1). Message queues represent unidirectional message paths. Consumers with one-of-n semantics concurrently pull messages. This interaction model is called point-to-point. The solution proposed in this paper focuses on this messaging mode.

![Figure 1: Point-to-point messaging](image)

#### Message Structure

A single message within a message-oriented, persistent communication infrastructure consists of multiple sections, two of which are optional: message properties and message body.

(i) The *message header* includes a number of predefined fields. These fields are used by the messaging system to decide how to process the message. They contain values that both clients and providers, use for identifying and routing messages.

(ii) The *message body* contains the business information (payload) of the message. This can be text, binary and in case of JMS a Java object that represents one of the five JMS message types: `TextMessage`, `MapMessage`, `BytesMessage`, `StreamMessage` or `ObjectMessage`.

(iii) JMS provides an additional section for supporting compatibility with other messaging systems: *message properties*. This part includes any arbitrary properties assigned in addition to the header fields. They are either application-specific or provider-specific.

#### Message Types

WebSphere MQ, the messaging system of main interest in this paper, uses four different message types in point-to-point mode, each of which may be chosen as an appropriate message type for the test application to be implemented [11]:

- **Datagram**: This is the standard message type. It is easy to handle and has fewer dependences than the other message types. Both sender and receiver are allowed to send independent datagram messages at any time.

- **Request**: A requestor sends a request message and waits for a reply message. Using a request message, at least two additional attributes have to be supplied within the message header: `replyToQueueManagerName` and `replyToQueueName`. These two parameters specify the return path, where the response is expected.

- **Reply**: An application, which receives a request message, has to respond with a reply message. A message correlation mechanism associates the two messages as one request/reply message pair [22]. The correlation id of the reply message has to be set to the message id of the correlating request message.

- **Report**: This message type is frequently used to inform an application about a messaging event, such as `confirm of arrival` and many others. A report can be requested either by the message type `datagram` or `request`. The corresponding answer may be released by an application, a queue manager or a message channel agent.

#### Object-Oriented Analysis

With the knowledge acquired so far, we can depict a simple messaging system as shown in Figure 2. A message queue represents a composition. Every queue manager owns at least one (in diagram: 1), usually more (in diagram: 1..*) message queues. The lifetime of a message queue is bound to that of a queue manager. A message queue does not own messages ("has-a" relationship). In object-oriented analysis, this situation is known as an aggregation. The lifetime of a message, however, is not bound to that of a queue: an already gathered message can survive as a text file. One or more producers respectively consumers are allowed to put messages into a queue or to get messages out of a queue. Nevertheless, allowing only one consumer may be appropriate, depending on the used messaging system [10, pp. 508]. The test messages to be implemented and the productive ones are both specialized elements of the generalized parent "message". This represents a "kind-of" relationship. A composition represents a stronger ownership than an aggregation. "Has-a" relationships simply are more flexible.
End-to-End Testing

End-to-end testing includes verifying the transactions through each application involved, from start to finish, assuring that all related processes occur correctly. The following active and passive components are of main interest: sender applications, message queues including the entire messaging system, receiver applications and – optionally – the legacy systems behind.

Enterprise Integration Patterns

The design patterns approach is a useful method for documenting an expert’s knowledge so that it can be readily understood and applied by others. A pattern represents a decision that must be made and the considerations that go into that decision [10]. Alexander [1, pp. 247] states: “Each pattern is a three-part rule, which expresses a relation between a certain context, a problem and a solution”. A pattern language is a web of related patterns in which each pattern leads to others, thus guiding one through the decision making process. Enterprise integration patterns offer a very important collection of proven solution concepts for integration issues, filling the gap between a high-level vision of the entire system and the present system implementation. Therefore, they are highly suitable as reference architecture for various approaches in systems and application integration.

3. GESTALT THEORY

Gestalt theory is a broadly interdisciplinary general theory, which provides a framework for a wide variety of psychological phenomena, processes, and applications. Gestalt psychologists were interested in complex phenomena: How do people perceive scenes and spaces; how do they solve complex problems; how do they relate different components of their experience, whatever the domain of the experience. Human beings are regarded as open systems in active interaction with their environment. Visual perception of human beings is characterized through the fast and automatic recognition of patterns and the changes in size, structure, color, movement and texture. Gestalt theory represents a family of psychological theories that have influenced many research areas, including visual design issues [5]. The German word Gestalt translates to “shape” or “form”. Gestalt theory first arose in 1890 as a reaction to atomism, the prevalent psychological theory of that time. Atomists believed the nature of things to be absolute and not dependent on context. According to Gestalt psychologists, however, context is very important in perception and human beings tend to organize visual perception into certain groups. Gestalt theory is, embedded in a greater context, part of systems thinking. Actually, the famous saying that “the whole is more than the sum of its parts” was coined by Gestalt psychologists. We tend to look at an overall image instead of looking at each visual element.

Basic Gestalt Principles

Gestalt theory is usually expressed as laws or principles. Many variants of Gestalt laws have been framed. According to Preece et al. [17] five basic Gestalt principles of perceptual grouping can be identified: proximity, similarity, closure, continuity and symmetry. Nevertheless, this number varies through different papers. Gestalt laws are used to describe how visual elements should be presented in order to achieve effective visual results.

Proximity refers to the distance of objects. Elements arranged closer together will be regarded as belonging together. Viewers will mentally organize closer elements into a coherent object, assuming that closely spaced elements are related.

Similarity is another grouping principle, which states that those elements that share common visual characteristics (qualities) such as shape, size, color, texture, value or orientation will be perceived as part of the same form.

Closure states that we tend to complete familiar figures. For example, we enclose spaces by completing contours and ignoring gaps in figures. We tend to see complete figures even when a part of the information is missing.

Continuity relates to our tendency to see patterns, concluding things belong together if they form a continuous pattern. Continuity does not interrupt visual connections between the objects. These connections are fundamental for preserving a unified visual statement.

Symmetry states that we tend to perceive shapes as figures based on their combined symmetrical forms, rather than on their individual asymmetric parts. A psychological sense of equilibrium or balance is usually achieved when visual equation is placed on each side of an axis.

Prägnanz or simplicity states that the most simple and stable interpretations are favored. When things are grasped as wholes, the minimal amount of energy is exerted in thinking. Koffka [13] proposed Prägnanz as the unifying law underlying the Gestalt principles of perceptual grouping explained earlier.

Example

Figure 3 shows the example of a HTML-page for accessing reports from four different units of interest: ES-Broadcast, ES-Receiving, STY-Broadcast and STY-Receiving. In this example, the Gestalt principles proximity, similarity, closure and symmetry were used to group sections into perceptually significant families. The three headings of the main report categories follow the principle of similarity. The four sub-headings form four imaginary columns, implementing the principle proximity. Symmetry was reached by mirroring the related departments ES.
and STY. The principle closure applies: the links to the detailed reports, the same names have been used in all four departments. The simple design includes the overall Gestalt principle of Prägnanz. Our example also shows that Gestalt principles are closely related or even overlap and it is very hard to distinguish between them.

ACF2-Sitemap

<table>
<thead>
<tr>
<th>DETAILED QUERIERS</th>
<th>ES-Broadcast</th>
<th>ES-Receiving</th>
<th>STY-Broadcast</th>
<th>STY-Receiving</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Users</td>
<td>New Users</td>
<td>New Users</td>
<td>New Users</td>
<td></td>
</tr>
<tr>
<td>Parameters</td>
<td>Parameters</td>
<td>Parameters</td>
<td>Parameters</td>
<td></td>
</tr>
<tr>
<td>Trids</td>
<td>Trids</td>
<td>Trids</td>
<td>Trids</td>
<td></td>
</tr>
<tr>
<td>Groups</td>
<td>Groups</td>
<td>Groups</td>
<td>Groups</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GRAPHICAL REPORTS</th>
<th>ES-Broadcast</th>
<th>ES-Receiving</th>
<th>STY-Broadcast</th>
<th>STY-Receiving</th>
</tr>
</thead>
<tbody>
<tr>
<td>Graphic</td>
<td>Graphic</td>
<td>Graphic</td>
<td>Graphic</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>METADATA</th>
<th>ES-Broadcast</th>
<th>ES-Receiving</th>
<th>STY-Broadcast</th>
<th>STY-Receiving</th>
</tr>
</thead>
<tbody>
<tr>
<td>FTP-File</td>
<td>FTP-File</td>
<td>FTP-File</td>
<td>FTP-File</td>
<td></td>
</tr>
</tbody>
</table>

Figure 3: Example – basic Gestalt principles applied

4. PLANNING ASPECTS

Messaging technology enables asynchronous, high-speed, program-to-program communication with reliable delivery. Despite its asynchronous communication model, messaging is frequently used as a time-critical base service for integrating distributed systems on application level. Let us take the example of an automotive plant: several applications exchange information via messaging to fulfill different business processes. The distributed systems involved are concerned with enterprise resource and production planning, automatic vehicle identification, logistic issues, robots and other production supporting activities. The messaging system used is WebSphere MQ in point-to-point mode. Several queues were set up on several queue managers to provide different business processes with unidirectional message paths.

Test Model

A system that fails will not adequately provide the services it was designed for. In our case, testing on network level (e.g. Simple Network Management Protocol) is not appropriate, since the main components of interest are part of the application level: producers, consumers and queues. Defining predictable failure situations on application level implies that failures occurring on the levels below may be caught, too [23]: network failures, crash failures, omission failures, timing failures, response failures and arbitrary failures. The following possible error situations should be checked within the time frame for which the test results are valid:

1) Queue managers are not available.
2) Message queues are constipated or have already reached their maximum limit of messages.
3) Message queues have accidentally got a wrong naming.
4) Sender and receiver applications are not working properly.
5) Messages are not able to arrive within the chosen time restrictions, e.g. because database transactions take too long.
6) The legacy systems are not able to answer just in time.

Modeling Methodology

Business process integration can be regarded as the highest level of abstraction, value of integration or manageability. It allows solution architects to design and manage information flow across a variety of applications. Using messaging middleware, business process integration requires a logical orchestration of the components involved. We distinguish between three layers of testing:

(i) The business layer focuses on analyzing and defining the business’ needs, independently of the solution used to meet a company’s requirements. This layer can be compared to a generic abstraction.

(ii) The logical layer specifies how business information can be exchanged in a structured way, following a number of rules. The purpose of the logical design is to refine the physical model in order to structure it for each business case and to parameterize the test application. This layer can be compared to specific requirements.

(iii) The physical layer delivers the messages and rules by means of syntax and code. Most of the business information involved is turned into data elements, of which the messages are composed. The entire set of our physical message paths to be tested is represented by M (Figure 4). Out of this M, subsets (M₁, M₂ … Mₙ) represent the business processes the message paths were set up for.

![Logical grouping of message paths](image)

Usability Requirements

One principle for visualizing information according to Shneiderman [20] is: “overview, zoom and filter, details on demand”. For example, a progress bar is required to support actions that take longer than about one second. The user target group wished the test results to be displayed in hierarchical layers representing the various distributed systems involved during testing, since they are already familiar with this kind of system landscape.

Discussing Messaging Issues

Testing asynchronous communication paths depends significantly on several criteria and conditions. What makes this approach so different and important is its focus on adaptations for the practical use in response to different requirements and constraints.
Send and Forget: Once a push operation (Figure 1) is complete, a sender does not have to wait for the answer. It does not even have to wait for the messaging system to deliver the message. Nevertheless, many loose-coupled implementations are time-critical for several reasons.

Delivery Sequence: Within a messaging system there is no guarantee for the point of time a message is delivered. Problems might occur if messages get out of sequence. For example, for the exchange of records between various databases, the messages have to be delivered in a strict first-in-first-out (FIFO) order to guarantee the consistency of the distributed applications. Using multiple receivers on a single queue, there is the risk that the first message in the queue takes longer to be processed than the second one. This is due to different time frames needed for database transactions. Practical tests have shown that a load-balanced system with a running receiver application for each instance can cause this problem. In this case, the messaging system was not able to manage multiple consumers in a coordinated way. To avoid running into these troubles, sometimes a turnaround might be necessary. A single reply queue offers a way out: only the receiver module of the test application will pull messages from this queue.

Frequency of Tests: In general, active testing adds burden to a system. As a requirement, the frequency of the tests applied must not produce side effects. Since a periodic heartbeat would both add burden and leave a gap of regularly untested periods, we consider event-based testing. Moreover, to avoid troubles caused by old test messages in a queue, an idempotent receiver would be appropriate. This idempotent receiver discards answers of test messages that do not belong to the current test run.

Time Constraint: Because of the characteristics of asynchronous communication we consider a well-defined time frame between sending out the test messages and receiving their answers. Our receiver application will pull them according to a strict FIFO order. After this time frame, missing answers of the test messages get classified as wrong. The value of this time constraint depends on the average time used to process productive messages and on the number of messages to be processed within a particular business process.

Expiry Time: Because of the time constraint discussed earlier, we consider an expiry time for test messages. After this expiry time, test messages are no longer valid and will therefore be discarded. This behavior contributes to reducing burden to the messaging system.

Transaction Certainty: As shown in Figure 2, a single queue allows several senders and receivers. In this case, the messaging system must be able to coordinate the messages to make sure senders do not overwrite each other’s messages. As a result, we have to ensure exactly-once delivery semantics [14] of the messages. Transactional queues, for instance, fulfill this need.

5. IMPLEMENTATION

Architecture
Our reflections on the architecture included issues such as reliability, persistence, scalability, usability and others. These considerations have led to a 3-layered-architecture, implemented with open source software. Different modules to separate data, logic and visualization were established:

(i) A data repository holds the entire data involved in testing: metadata and the test results. Metadata, for example, defines the message paths to be tested. The storage of test results has two advantages. Firstly, the test results can be analyzed at a later time, e.g. for statistical reasons, and secondly, analyzing does not require particular tools.

(ii) The test program, coded in Java, contains the business logic of the test application. It utilizes the API of the message queuing system for sending and receiving the test messages and their answers, as well as a JDBC-compliant data source adapter to handle data-accessing issues. Together with the classes and methods for visualizing the test results, this java program resides on a Tomcat Web container and can easily be accessed via a browser.

(iii) For visualizing the test results, Java Server Pages (JSP), Tag Libraries (TagLibs) and pure java code are used. Linear data, as stored in the data repository, had to be converted into hierarchical layers for displaying the different distributed systems involved. This was simply accomplished by using HTML. Depending on a single parameter, which indicates the current business process, the proper system landscape is loaded at the end of each test run. This feature refers to the issue scalability. A new business process to be tested can be implemented by simply applying a new parameter, the set of message paths to be tested and a new JSP-page to visualize the system landscape.

Message Structure
So far, all the productive messages belong to the message type datagram. An additional user-defined header within the payload area is used to identify different messages for different business processes. To guarantee compatibility, we decided in favor of the message type datagram also for the test messages. In fact, a test message implements a specialization of the generalized object message (Figure 2). To meet the requirements, the content of the test message was planned very carefully. Several attributes and parameters were applied:

1) Unique message sequences for each test run to distinguish between multiple clusters of test messages. This allows identifying the test results that belong together.
2) The return address (queue manager, queue) is necessary to specify the return path for the expected answers.
3) Indicators for the business processes the message paths are used for. As a result, a logical grouping of the physical message is possible (Figure 4).
4) A timestamp, providing the respective send-date.
5) A well-defined time constraint. Receivers are expected to reply within this frame.
6) Additional properties for displaying the test results, such as direction indicators.
7) An activity flag that indicates at runtime, whether a message path is allowed to be tested or not.
8) An area of about 200 characters to hold an optional answer string of the systems behind the receiver applications.

As an optional requirement, the integrated legacy systems are expected to report brief status information for the user. This is a novel approach that turns out to be very helpful in interpreting the test results. As a result, the health status of several distributed systems may be observed in assistance of this test tool.
Patterns Applied
During implementation, several enterprise integration patterns were applied. A content-based router [6] offers a generic approach to the question of how to handle a situation in which the implementation of a single logical function is spread across multiple physical systems. This pattern was applied for distributing the test messages. Each message is being routed to the correct recipient, based on the content of the message (address). The routing information is gathered from the data repository and represented by attributes of the result set. The necessary address information is provided by the attributes QM and QUEUE, which represent queue manager and queue. Based on these attributes, the content-based router decides where to send the messages. The return path is defined by the attributes REPLYTOQM and REPLYQUEUE. Despite this enhanced feature, there is only one point of configuration for the entire application: just one table within the data repository.

The aggregator pattern shows a way of how to combine the results of individual, yet related messages so that they can be processed as a whole. The original idea of an aggregator is to use a stateful filter for collecting and storing individual messages until a complete set of related messages was received. Subsequently, the aggregator publishes a single message distilled from the individual messages. We implemented a solution derived from the aggregator pattern. Instead of publishing a single message after collecting all the individual messages, we update our data repository holding all the sent messages and change their status from “transmitted” to “received”. This status information is necessary for visualizing the test results.

The part for accessing the data repository was implemented by applying classical Java patterns such as the factory pattern. As a result of our intensive work with patterns, the author published his own ideas for a process-related test message at EuroPLoP 2004 [8].

6. VISUALIZING THE TEST RESULTS

Gestalt Principles Applied
Many of the basic Gestalt principles introduced earlier were considered during implementation. Each business process is implemented on a separate page. Together with a mechanism that passes on parameters dynamically, scalability is guaranteed. This method refers to the unifying Gestalt principle Prägnanz: The overall appearance for each business process is simple and clear. Similarity was applied by using the same color for shapes representing systems that belong together in a logical sense, for example, the shapes of SAM and AVI, representing two distributed systems: production planning system and automatic vehicle identification. The Gestalt principle proximity was applied by contracting the symbols that represent a pair of queues. In the context of a bidirectional information exchange, sender and receiver queues belong together. Continuity was applied by adapting the appearance of this web application to the company’s intranet system, this way persuading the user’s eye to complete an overall image.

Semantic Aspects
Human beings combine perceived patterns with cognitive knowledge. When the appearance of an object is perceptually obvious, it is easy to know how to interact with it. The colors of the traffic lights are a well-known example: we have learned a red traffic light means danger, whereas a green light indicates the opposite meaning. According to this example, colors are used in the semantics of the test results. The different operational states of the tested message paths are displayed as arrows in different colors. Arrows were used to indicate the unidirectional message flow within a queue.

<table>
<thead>
<tr>
<th>Color</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Green</td>
<td>Queue is working properly</td>
</tr>
<tr>
<td>Red</td>
<td>Problems occurred in the queue</td>
</tr>
<tr>
<td>Black</td>
<td>Not tested yet</td>
</tr>
</tbody>
</table>

Table 1: Semantics of the colors used

7. RELATED WORK

In this section we briefly introduce related work in the area of patterns, messaging and Gestalt theory. Alexander pioneered the concept of patterns and pattern languages in his books A Timeless Way of Building [1] and A Pattern Language [2]. An excellent resource for patterns is Hillside [9]. End-to-end arguments in system design are discussed comprehensively by Saltzer et al. [19]. Messaging systems are widely explored through different papers [7], [14], [22]. Human-Computer Interaction (HCI) [15], like other multidisciplinary fields, borrows techniques from its component disciplines and has to determine how they relate to each other. Gestalt theory – which came up from psychology – has been discussed in many papers [3], [5], [12], [13], and [24]. Chang et al. [5] evaluated eleven key laws of Gestalt theory for computer screen design. However, regularly coming across web sites that ignore these basic design principles encouraged me to focus on Gestalt principles of perceptual grouping. The author is currently working on a draft paper, mining and specifying the basic Gestalt principles as a pattern language for visual information design. Well-established categories in linguistics, namely morphology, syntax, semantics, and pragmatics and their relationship to the proposed pattern language will be discussed. The resulting paper will most likely be presented at one of the next PLoP conferences.

8. CONCLUSIONS

This paper spans a range of subjects, from communication technology over software development to Gestalt theory as part of systems thinking, which means putting something into the context of a larger whole. In the sense of Prägnanz, a good whole is characterized by simplicity and stability. We have discussed point-to-point messaging, end-to-end testing and basic Gestalt principles of perceptual grouping in detail. Design patterns, namely enterprise integration patterns, complement these main topics. We presented a comprehensive tool for monitoring a messaging infrastructure on application level by means of a logical orchestration of its metadata. Key features are a clear separation of layers, an active, process-related monitoring mechanism with a customizable view for each business process. Gathering status information from the legacy systems behind and displaying them on demand completes this end-to-end test on application level. The possibility of storing all the test messages passing through the end-to-end connection increases system reliability and provides great flexibility in reporting and analysis. Displaying the test results in a manner that incorporates Gestalt principles of perceptual grouping is a novel approach and was well worth the effort to get visualization into the right place.
9. REFERENCES