Multidomain Event-Based Middleware using Decentralised Information Flow Control

Extended abstract for DEBS 2009 PhD Workshop

Ioannis Papagiannis
ip108@doc.ic.ac.uk
Imperial College London
Supervisor: Peter Pietzuch

1. EVENT-BASED MIDDLEWARE

Event-based middleware systems have tried to interoperate with various platforms and languages. As this goal became reality for the most important and widespread platforms, the adoption of these systems begun. IBM’s WebSphere MQ, Sun’s JMS together with non event based architectures like CORBA, were industrial solutions that soon functioned as the interconnection mechanism behind server platforms like WebSphere and J2EE. Even though the interoperability achieved by these systems is impressive, most of the time they assume that all the code that is executed is equal in terms of quality, auditing and debugging.

This assumption holds for most parts of an event-based system but it is not always true. Features such as client provided operators, event transformation routines and multi-node systems under different administrative domains result in deployments where not all the code can be equally trusted. Moreover, as systems become more complicated, they are prone to bugs that can affect the security of the whole platform. In scenarios like these, code testing may vary greatly from detailed to completely unavailable, yet all processing components have to access the events that they operate on.

As seen in the example of Figure 1, a third party instantiates in the system an event processor which transforms the name that appears on certain events. Suppose that this event stream contains medical information and the patient’s personal details must be protected from disclosure. As a result, this processor component will have to access the sensitive name part of the events, yet it must not communicate it to untrusted receivers. If no guarantees are given, then the system’s security will be directly dependant upon this component’s behaviour: the component can choose to communicate the name to any receiver that it wishes. The component may as well be part of the event-based system’s internal codebase; protection is then useful in limiting the effects of possibly exploitable bugs.

In order to provide security in a multidomain application like the above, two fundamental properties must be guaranteed by the system: data confidentiality and integrity.

- A data producer must have the ability to confine the data that it considers sensitive from disclosure. Partially trusted code must not be able to leak other producers’ sensitive data without their consent.
- Partially trusted code must not be able to generate data and forge them so that they appear to have been produced by other producers. Any subscriber must be able to request only data that have been generated by trusted sources.

If these two guarantees are provided, the system can use code of different trust levels and operate effectively.

2. DIFC

One promising technique of achieving the above guarantees is the tracking of information flow inside the event-based system. By capturing at all times which components have observed sensitive information, dynamic restrictions can be imposed that will contain the further propagation of sensitive information. Decentralised Information Flow Control [6] (DIFC), introduced as an extension to traditional Information Flow Control [2] (IFC) systems, is such a flow tracking paradigm. Unlike IFC, DIFC allows the system’s components themselves to take part in the creation of security policies that the system must enforce. Thus, the creation of security policies becomes decentralised and every application can use this mechanism to enforce any self-generated policies. Since its introduction, DIFC has been...
used as a paradigm to build a new generation of Operating Systems [4, 7] or existing Operating System’s extensions [5].

Data are labelled in various classes and each system’s component is only allowed to access them if it has a compatible clearance. Components can alter their clearance according to a set of privileges and thus they are able to receive more or less tainted data. Components’ output is also affected by their clearance: every component is forced to output data that inherit the labels of its inputs. As a result, the cleared components can receive the information they require, yet their output will only be perceivable by other cleared components. In the end, the data will stay in the system, unless a component with declassification privileges declassifies them. By allowing components to operate using only clearance, the system guarantees that they will not communicate their output to other untrusted components.

3. RESEARCH CHALLENGES

The first challenge in this work will be to achieve containment. As described, different code will run with different privileges that describe the security properties of the code’s output. These privileges will be the input for all decisions about the data’s dissemination and the dissemination flow implied must never be violated. At the same time, most event based middleware systems are in fact developed in high level, type safe languages such as Java and C#. Even though type safety is a significant advantage when considering containment, many problems appear when we want to confine application components that execute concurrently.

First of all, partially trusted code must be contained from accessing the system’s main resources, such as the file system, the network etc. Fortunately, both Java and C# offer functionality that can be used for this purpose. At the same time, partially trusted code of one provider must also be contained from communicating with other partially trusted code of other providers. This implies that our solution requires interception of all communications between two components running on a single runtime, a goal that has been a separate research direction on its own [1].

The second challenge is performance. DIFC is a runtime mechanism; any component’s communication involves checks that must verify if the communication is “safe” in terms of information flow. These checks are mainly comparisons between labels, and their effect on performance must be minimised. Similarly, the confinement mechanism presents another source of runtime overhead. In languages such as C# where the runtime offers cross-application confinement, the overhead of crossing the cross-application boundary is significant. As a result, the isolation enforcement mechanism not only must be safe but it must also be efficient when it comes to authorised data dissemination.

The third challenge is to make this mechanism easy to use. Initial research in this area [3] shows that DIFC is a paradigm that is prone to mistakes. The number of labels required, even for relatively simple applications, can easily overwhelm the programmer. These errors may result in system policies different than the ones expected and can lead to violations of the intended information flow. Moreover, the required flow of information is often unknown when the programmer develops a component, only to be defined by the system’s administrator at deployment time. In other words, information flow policies should not be hard-encoded in the components but they should be decided on a per-context basis instead. For the above reasons, a higher level information flow policy language will improve the usability of the system and will allow less experienced users to use it effectively.

4. CONCLUSION

To sum up, the main challenge that my research will try to investigate is how we can build event-based middleware systems and guarantee their security properties in situations where the level of trust varies for each component. To achieve this goal, the focus of this research will initially be the creation of a DIFC substrate that will offer the ability to develop arbitrary DIFC-enabled event-based applications. For this substrate, a new comprehensive DIFC model is needed that encapsulates the experiences of the systems that have implemented it before [4, 7, 5]. These systems have improved the initial DIFC model [6], yet there is still work to be done in order to make this mechanism suitable for event-based systems. Additionally, the containment mechanism that will be used must be described in detail aiming to make it hard to circumvent and yet relatively lightweight.

Using this DIFC-enabled substrate, a classic publish/subscribe system will be developed in order to explore the possibilities and the implications that DIFC introduces to its operation. This system will be an example of a multidomain event-based system: different event processing units will be assumed to be of different trust and each one of them will try to protect its sensitive data from disclosure. This system will be used to study DIFC’s performance impact and will also show how a system produced by a single vendor can benefit from avoiding the implications of possible bugs.

Finally, I will try to research the possible ways to make the DIFC paradigm easier to use. For this reason, a policy language like the one described above will be adopted in order to decouple the components processing and the definition of the expected information flow.

5. REFERENCES