Evaluation of the Applicability of the OSGi Service Platform to Future In-Vehicle Embedded Systems

Irina Astrova, Ahto Kalja
Institute of Cybernetics
Tallinn University of Technology
Tallinn, Estonia
irina@cs.ioc.ee, ahto@cs.ioc.ee

Arne Koschel, Roman Roelofsen
Faculty IV, Dept. for Computer Science
Applied University of Sciences and Arts
Hannover, Germany
akoschel@acm.org, roman.roelofsen@googlemail.com

***Abstract*—One promising market for embedded systems is the automotive industry. For example, engines, dashboards, wipers, lights, doors, windows, seats, mirrors, radios, CDs, DVDs, and hand-free phones are being controlled by in-vehicle embedded systems. This paper evaluates the applicability of the OSGi Service Platform to future in-vehicle embedded systems. One specific application area, which can greatly benefit from this evaluation, is Web services, where a number of future scenarios have not been supported by the OSGi Service Platform yet. This is exemplified by two example scenarios, viz., a car tracking service and an advertising service. Based on these example scenarios, a number of requirements that should be met by the OSGi Service Platform are identified, viz., dynamic availability, versioning, persistence, composition, remote management, platform independence, security, and distribution. In addition to the identification of the requirements, another contribution of this paper is the evaluation of the OSGi Service Platform against these requirements. This evaluation will help to extend the OSGi Service Platform to be applicable to future in-vehicle embedded systems.**

Keywords—OSGi Service Platform, OSGi Framework, in-vehicle embedded systems, Web services, car tracking service, advertising service

I. INTRODUCTION

"Nowadays, for any activity in our everyday life, we are likely to use products and services, whose behavior is governed by computer-based systems also called *embedded systems*" [4]. Embedded systems constitute the biggest sector in the market today. "Of the 9 billion processors manufactured in 2005, less than 2% became the brains off new PCs, Macs, and Unix workstations. The other 8.8 billion went into embedded systems" [2].

This market trend also affects the automotive industry, one of the largest economies in the world. Over the last two decades, there has been an exponential increase in the number of computer-based systems embedded in vehicles also called *in-vehicle embedded systems* (or *automotive embedded systems*) [4].

In-vehicle embedded systems are currently used for navigation, climate control, adaptive control, traction control, stabilization control and active safety. In the future, they will also be used for remote diagnostics of a vehicle, access to the Internet and audio/video entertainment. The cost of in-vehicle embedded systems constitutes more than 25% of the total cost of a vehicle today [11]. In 2010, in-vehicle embedded systems will account for 40% of a vehicle's content [3].

The remainder of this paper is organized as follows. At first, the paper describes example scenarios of using Web services in future in-vehicle embedded systems, viz., a car tracking service and an advertising service. Next, the paper lists requirements driven by the example scenarios, viz., dynamic availability, versioning, persistence, composition, remote management, platform independence, security, isolation, communication, and distribution. Finally, the paper evaluates the OSGi Service Platform to see if this platform can meet the requirements of the example scenarios.

II. MOTIVATION

The motivation that leads us to evaluate the applicability of the OSGi (Open Services Gateway initiative) Service Platform [8] to future in-vehicle embedded systems stems from the following facts:

1. The OSGi Service Platform originally targeted gateways (as can be deduced from the platform name). However, the platform has been adapted to many other domains, including vehicles.
2. Vehicle manufactures (producing around 70 million cars per year [20]) have showed interest in the OSGi Service Platform. Simple evidence of this fact is that Automotive Multimedia Interface Collaboration (representing major vehicle manufactures) has joined the OSGi Alliance [18].
3. Vehicles are a promising market for embedded systems. For example, engines, dashboards, wipers, lights, doors, windows, seats, mirrors, radios, CDs, DVDs, and hand-free phones are controlled by embedded systems.
4. More than 80% of all new innovations in vehicles are based on embedded systems [19]. For example, when an accident causes an airbag to inflate in a Cadillac, an embedded system in the car emits a signal for the global positioning system (GPS) service to get the car's position and then communicates with the driver's cell phone to send the car's position to the rescue service [4].

 ISBN: 978-1-61208-105-2

III. EMBEDDED SYSTEMS

An *embedded system* is "a micro-processed device, thus programmable, which uses its processing power for a specific purpose" [1]. It typically consists of memory (such as RAM, EPROM, ROM or flash memory), a processor (such as Intel x86, PowerPC or ARM), a clock and an input/output device (see Figure 1).

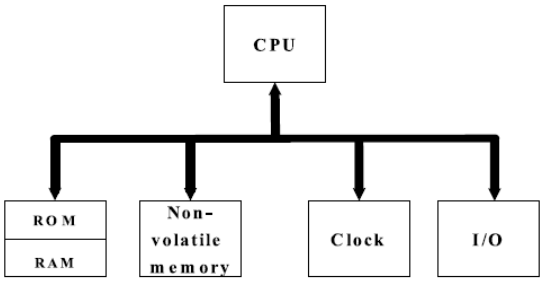

Figure 1. Embedded system [1]

Because of small memory footprints, embedded systems are also called *small-memory devices*. However, the size of software embedded in vehicles has been growing. For example, in 1980 and 2000, a Peugeot contained 1.1 KB and 2 MB, respectively [4]. In 2010, some vehicles may contain 100 million lines of code [3].

The growing size of embedded software also raises a question of its quality. For example, in 2003, 49.2% of all car breakdowns in Germany were due to bugs in embedded software [4].

IV. WEB SERVICES

A *Web service* is "a software system designed to support interoperable computer-to-computer interaction over a network" [10].

The market trend towards embedded systems gives rise to the idea of using Web services in embedded systems. However, this is a challenging task because "embedded systems rarely have enough memory and processing power to run Web services" [6].

V. OSGI SERVICE PLATFORM

The OSGi Service Platform [8] is a Java platform that has arisen in the context of embedded systems. The platform is freely available and constantly developed by the OSGi Alliance [8].

There are several commercial and non-commercial implementations of the OSGi Service Platform, including Eclipse Equinox [30], Apache Felix [31], Knopflerfish [32], and ProSyst's mBedded server [33]. Well-known applications that are based on the platform include the Eclipse IDE and Apache Service Mix [34].

The core of the OSGi Service Platform is the OSGi Framework. This framework simplifies the development and deployment of extensible applications also called *bundles*, by decoupling the bundle's specification from its implementation. This means that a bundle is accessed by the framework through an interface, which is by definition separate from the bundle's implementation. This separation enables changing the bundle's implementation without changing the environment and other bundles.

The OSGi Framework makes it possible to run multiple applications simultaneously within a single Java Virtual Machine (JVM), by dividing applications into bundles that can be loaded at runtime and also removed. For communication within the JVM, the framework provides a service registry to register services, so that services can be found and used by other bundles.

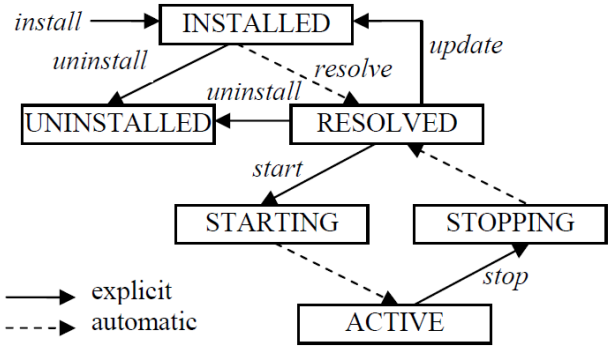

Figure 2. Life cycle of bundles in the OSGi Framework [9]

A bundle has a well-defined life cycle (see Figure 2) and its own context also called a class loader. It can be in one of the following states:

1. *Installed.* The bundle has been installed. After this, the bundle will be moved into the resolved state.
2. *Resolved.* All classes that the bundle needs have been loaded. This state indicates that the bundle is either ready to be started or has stopped.
3. *Starting.* The bundle is being started. After this, the bundle will be moved into the active state.
4. *Active.* The bundle has been activated and is running. The bundle's functionality is provided and its services are exposed to other bundles registered in the service registry.
5. *Stopping.* The bundle is being stopped. After this, the bundle will be moved into the resolved state.
6. *Uninstalled.* The bundle has been uninstalled.

VI. EXAMPLE SCENARIOS

To identify the requirements for the OSGi Service Platform in the vehicle domain, we considered the following examples of using Web services in embedded systems:

1. Car tracking service (see Figure 3a).
2. Advertising service (see Figure 3b).

 ISBN: 978-1-61208-105-2

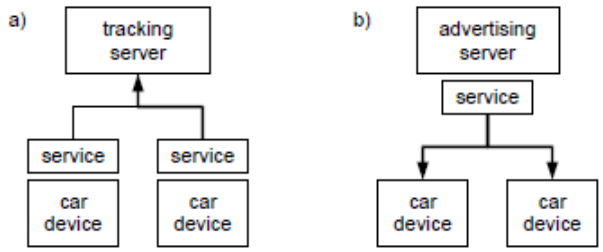

Figure 3. Example scenarios: (a) car tracking service; and (b) advertising service [5]

These examples were derived from vehicle manufacture opinions about the future direction of using Web services in embedded systems.

A. Car Tracking Service

The car tracking service will be used by the car rental company to get the car's position. Installation of this service will be initiated either by the driver or by the car rental company.

B. Advertising Service

The advertising service will be used by the driver (e.g., waiting in a traffic jam) to get advertising information. This service will be installed by the local advertising server just in time as the car enters a local hot-spot network.

VII. Requirements of Example Scenarios

We identified the following requirements for the OSGi Service Platform in the vehicle domain:

1. Dynamic availability.
2. Versioning.
3. Persistence.
4. Composition.
5. Remote management.
6. Platform independence.
7. Security.
8. Isolation.
9. Communication.
10. Distribution.

These requirements were driven by the example scenarios.

A. Dynamic Availability

Since embedded systems have small memory footprints, it is important to keep the memory footprint at runtime as low as possible. Therefore, it should be possible to install services on a running system only when they are really needed and uninstall them afterwards (e.g., when they are no longer needed) without requiring the system to be restarted, as this would also affect other services and temporarily stop them from running. In addition, since services may change over time, it should be possible to update them at runtime. However, only the smallest possible set of services should be affected by that update.

B. Versioning

Since services can be updated at runtime, it should be possible to reflect that update as new versions of services (e.g., by assigning version numbers to services).

C. Persistence

It should be possible to make services persistent. For example, once installed, the car tracking service can be saved for later reuse. This is by contrast to the advertising service that will usually be removed just in time as the car leaves the local hot-spot network.

D. Composition

An application can be composed of multiple services in order for the running system to install services only when they are really needed. Again, this helps to keep the memory footprint at runtime as low as possible. Therefore, it should be possible to deploy a composite application. However, the order in which services will be installed (e.g., to add new functionality to the running system) should not be fixed.

E. Remote Management

Since services can be installed, uninstalled and updated by external systems, it should be possible to manage the life cycle of services from the outside world.

F. Platform Independence

Since services can be installed by external systems, they cannot know in advance all platforms on which they will run. Therefore, it should be possible to run services unchanged on multiple platforms (including different hardware platforms and operating systems).

G. Security

Security is important because networks represent a potential avenue of attacks to any embedded system connected to them. Since networks allow embedded systems to communicate with external systems, networks can potentially serve as a way to break into embedded systems, enabling malicious (or buggy) services to steal resources or data. As a consequence, communication over a network raises many security issues. These issues become especially important in distributed environments, where code is downloaded across networks but executed locally as is done, e.g., with the advertising service. Because the advertising service is automatically downloaded when the car enters a local hot-spot network, it is likely that the driver will encounter code from untrusted sources.

Without security, communication of embedded systems with external systems would be a convenient way to distribute malicious services. Therefore, this communication will require the use of security techniques and technologies, from firewalls to cryptography and identity certification that are necessary to prevent services to be downloaded from untrusted sources and to prevent unauthorized remote life cycle management as well as unauthorized access to resources and data. For example, an embedded system can store the phone book from the driver's cell phone. This information should be protected from access by the

 ISBN: 978-1-61208-105-2

advertising service (e.g., to ensure that the information will not be accidentally or maliciously modified).

H. Isolation

Different services such as the car tracking service and the advertising service can be deployed on the same embedded system. Therefore, it should be possible to prevent services from interfering each other (e.g., by isolating them).

I. Communication

While some services can live in isolation, others may need to work together to combine their functionality. Therefore, it should be possible to enable communication between services. For example, the car tracking service can communicate with the GPS service to get the car's position.

The main challenge here is to preserve security of information from, to, and within embedded systems. For example, the car's position can be transmitted over an insecure network such as the Internet. Therefore, it should be possible to encrypt this information.

J. Distribution

Services can be distributed across a network for the purpose of high availability, performance, or just due to their popularity. Therefore, it should be possible to deploy services in distributed environments, which usually involve versioning and communication.

VIII. EVALUATING THE OSGI SERVICE PLATFORM

We evaluated the OSGi Service Platform against the requirements of the example scenarios. Table I summarizes the results of our evaluation.

TABLE I. SUMMARY OF EVALUATION RESULTS. 'YES' – REQUIREMENT IS FULLY MET. 'YES/NO' – REQUIREMENT IS PARTIALLY MET. 'NO' – REQUIREMENT IS NOT MET AT ALL

Requirement	**Is Requirement Met?**
Dynamic availability	Yes
Versioning	Yes
Persistence	Yes
Composition	Yes
Remote management	Yes
Platform independence	Yes
Security	Yes/No
Isolation	Yes
Communication	Yes/No
Distribution	Yes/No

A. Dynamic Availability

Applications represented as bundles for deployment can be installed, uninstalled and updated dynamically (i.e., at runtime) without requiring a restart of the OSGi Framework.

B. Versioning

Bundles export services by registering them in the service registry. During this registration, additional information (including version numbers) can be assigned to the services.

The OSGi Framework goes even further in supporting versioning semantics. Versions can be assigned to export packages as well.

C. Persistence

Bundles are stored in the persistent storage of the OSGi Framework and remain there until they are uninstalled. Whenever the framework is restarted, bundles will be set to the same state they had just before the framework shut down.

D. Composition

An application can be represented as a single bundle. But it can also be composed of multiple bundles.

E. Remote Management

Bundles can be installed, uninstalled and updated remotely (i.e., from the outside world) without requiring a restart of the OSGi Framework.

F. Platform independence

The OSGi Framework is built on top of a JVM. Therefore, bundles can run on any platform that hosts the JVM.

G. Security

The OSGi Framework is focused on protecting embedded systems from code downloaded across a network from untrusted sources. When a bundle requests access to a particular resource, the framework grants the bundle access to that resource if and only if such access is a privilege associated with that bundle. Access control is based on digital signing, which authenticates the signer and ensures that a bundle's content is not modified after it has been digitally signed by the principal. Digital signing is based on a public key cryptography.

Not only are privileges granted to code and signers, but they are also granted to principals on whose behalf code is being executed. For example, a bundle can be granted the permission to manage the life cycle of other bundles that are digitally signed by the principal.

Although the OSGi Framework imposes strict access control on what code can and cannot do, the framework is not completely protected against damages from interference of malicious bundles. For example, malicious bundles can modify shared objects such as static variables, interned strings and `java.lang.Class` instances. This modification can affect other bundles running in the same JVM. Malicious bundles can even perform denial of service attacks against resources such as memory and a processor [12].

H. Isolation

Bundles are isolated from each other by class loaders.

 ISBN: 978-1-61208-105-2

I. Communication

Bundles can communicate with each other if they are loaded by the same class loader. When they are loaded by different class loaders, their communication is possible using a shared parent class loader only. However, this class loader is not part of the standard JVM and therefore requires an additional custom solution.

J. Distribution

To reduce the memory footprint at runtime (which is important for embedded systems) and to help avoid library redundancies, all bundles run in the same JVM. Thus, they can communicate with each other within a local framework only.

The OSGi Framework supports "rudimentary" distribution through Universal Plug and Play (UPnP) [17]. UPnP is a collection of networking protocols – e.g., TCP, UDP, IP, and HTTP – that allows networked devices to automatically communicate with each another. In particular, once a device is plugged into a network, it can access other devices connected to the network whereas other devices can access it.

IX. Related Work

In our previous work [21], we evaluated the Java Platform against the requirements of the example scenarios. The OSGi Service Platform is based on the Java Platform. But it addresses some of the weaknesses of the Java Platform; e.g., dynamic availability [7] (i.e., the ability of services to come and go at any point in time), versioning, persistence, and security. Thus, the OSGi Service Platform is more advanced than the Java Platform.

The OSGi Alliance has the Vehicle Expert Group, which aims to gather vehicle-specific requirements. These requirements are then used to tailor and extend the OSGi Service Platform. The OSGi Framework remains unchanged to provide upward compatibility.

X. Conclusion and Future Work

The OSGi Service Platform is a promising platform for future in-vehicle embedded systems and its concepts (such as bundles, class loaders and a JVM) help to meet many of the requirements of the example scenarios, viz., the car tracking service and the advertising service. However, limited support of security and distribution can be a severe hindrance for the use of the OSGi Service Platform in future in-vehicle embedded systems.

A. Security

The OSGi Framework has 25 security holes [23]. While 17 of them are due to weaknesses of the framework itself, other 8 are due to weaknesses of the JVM and the isolation mechanism provided by class loaders. However, this problem is being vanished over the years as a number of efforts – e.g., KaffeOS [24], Luna [25], and I-JVM [26] – have been made to patch the security holes that the JVM and class loaders leave open.

Since more and more in-vehicle embedded systems will communicate with external systems through an insecure network such as the Internet, security becomes important. However, proven solutions such as SSL/TLS for transport security and even recently issued standard such as WS-Security [22] for Web services require a lot of processing power and can disrupt the operation of embedded systems. Moreover, security increases the delay, jitter and deviation time [13].

B. Distribution

Limited support of distribution also becomes a less severe problem as a number of efforts – e.g., R-OSGi [14], Distributed OSGi [17], IBM Lotus Expeditor [27], Eclipse Communication Framework [15], Newton Framework [28], and Apache CXF [16] – have been made to add the distributed capability to the OSGi Framework, thus enabling bundles running in one framework to communicate with bundles running in another, potentially remote, framework.

C. Future Work

In the future, we will evaluate another Java platform such as JAIN SLEE [29] against the requirements of the example scenarios. This evaluation will help to extend JAIN SLEE to be applicable to future in-vehicle embedded systems.

Acknowledgment

Irina Astrova's and Ahto Kalja's work was supported by the Estonian Centre of Excellence in Computer Science (EXCS) funded mainly by the European Regional Development Fund (ERDF).

References

[1] G. Machado, F. Siqueira, R. Mittmann, and C. Vieira e Vieira. Embedded Systems Integration Using Web Services, Proceedings of the International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, April 2006.

[2] M. Barr. Embedded Systems Glossary, last accessed: June 2010, http://www.netrino.com/Publications/Glossary

[3] Automotive Industry, last accessed: June 2010, www.windriver.com/solutions/automotive

[4] Automotive Embedded Systems Handbook, eds. N. Navet and F. Simonot-Lion, CRC Press, 2009

[5] R. Roelofsen, D. Bosschaert, V. Ahlers, A. Koschel, and I. Astrova. Think Large, Act Small: An Approach to Web Services for Embedded Systems Based on the OSGi Framework, Lect. Notes in Business Inform. Process. 53, eds. by J.-H. Morin, J. Ralyté, and M. Snene, Springer, Berlin, 2010, pp. 239-253

[6] M. Barr and A. Massa. Programming Embedded Systems, 2nd edition. O'Reilly, CA, USA, 2007

[7] H. Cervantes and R. Hall. Automating Service Dependency Management in a Service-Oriented Component Model, Proceedings of the 6th Workshop on Component-Based Software Engineering, May 2003

[8] OSGi Alliance: OSGi – The Dynamic Module System for Java, last accessed: June 2010, http://www.osgi.org

[9] M. Persson. Resource and Service Registration and Lookup in Cooperating Embedded Systems, acc. 6/2010, http://www2.hh.se/staff/tola/cooperating_embedded_systems/papers/magnus_persson_final.pdf

 ISBN: 978-1-61208-105-2

[10] World Wide Web Consortium: Web Services Activity, last accessed: June 2010, http://www.w3.org/2002/ws

[11] H. Kopetz. The time-triggered architecture, Proceedings of the 1st International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC), Kyoto, Japan, April 1998, pp. 22-31

[12] N. Geoffray, G. Thomas, B. Folliot, and C. Clement. Towards a New Isolation Abstraction for OSGi, Proceedings of the 1st Workshop on Isolation and Integration in Embedded Systems, 2008

[13] M. Shopov, H. Matev, and G. Spasov. Evaluation of Web Services Implementation for ARM-based Embedded System, Proceedings of ELECTRONICS'07, Sozopol, Bulgaria, September 2007, pp. 79-84, ISBN: 1313-1842

[14] Swiss Federal Institute of Technology (ETH) Zurich: R-OSGi pages, last accessed: June 2010, http://r-osgi.sourceforge.net

[15] Eclipse Foundation. Eclipse Communication Framework, last accessed: June 2010, http://www.eclipse.org/ecf

[16] Apache Software Foundation: CXF pages, last accessed: June 2010, http://cxf.apache.org

[17] R. Santoso. Initial Idea: Distributed OSGi Through Web Services, last accessed: June 2010, http://www.dosgi.com , http://www.dosgi.com/index/39-distributed-osgi-webservices-articles-category/55-initial-idea-distributed-osgi-through-web-services.pdf

[18] T. Honkanen. OSGi — Open Services Gateway initiative, last accessed: June 2010, http://www.automationit.hut.fi/julkaisut/documents/seminars/sem-s01/honkanen.pdf

[19] K. Hackbarth. OSGi — Service-Delivery-Platform for Car Telematics and Infotainment Systems, Advanced Microsystems for Automotive Applications, Springer, pp. 497 – 507, 2003

[20] H. Kopetz. The time-triggered approach to real-time system design. In Predictably Dependable Computing Systems, eds. B. Randell, J.-C. Laprie,H.Kopetz, B. Littlewood. Springer-Verlag, New York, 1995

[21] R. Roelofsen, A. Koschel, and I. Astrova. Evaluation of Life Cycle Functionality of Java Platform, Proceedings of the 14th WSEAS International Conference on COMPUTERS, Corfu, Greece, July 2010, pp. 69-74

[22] Web Services Interoperability Organization. Basic Security Profile Version 1.0. last accessed: June 2010, http://www.ws-i.org/Profiles/BasicSecurityProfile-1.0.html

[23] P. Parrend and S. Frenot. Security benchmarks of OSGi platforms: Toward hardened OSGi. Software: Practice and Experience, 39(5):471-499, April 2009

[24] G. Back, W. Hsieh, and J. Lepreau. Processes in KaffeOS: Isolation, resource management and sharing in Java. Proceedings of the Syposium on Operating Systems Design and Implementation, San Diego, USA, October 2000

[25] C. Hawblitzel and T. Eicken. Luna: A flexible Java protection system, SIGOPS Operating Systems Review, 36(SI):391-403, 2002

[26] N. Geoffray, G. Thomas, G. Muller, P. Parrend, S. Frénot and B. Folliot. I-JVM: a Java Virtual Machine for Component Isolation in OSGi, Proceedings of 39th IEEE/IFIP Conference on Dependable Systems and Networks (DSN), Lisbon, Portugal, 2009

[27] IBM, Lotus Expeditor – Product and DeveloperWorks pages, last accessed: June 2010, http://www-01.ibm.com/software/lotus/products/expeditor/, http://www.ibm.com/developerworks/lotus/products/expeditor/

[28] Paremus Limited. Newton 1.5 Developer Guide, last accessed: June 2010, http://newton.codecauldron.org, http://newton.codecauldron.org/site/ newton-1.5-DEV-developer-guide.pdf

[29] JCP-JSR204, JSR 240: JAIN SLEE (JSLEE) v1.1 Spec., last accessed: June 2010, http://www.jcp.org/en/jsr/detail?id=240

[30] J. McAffer, P. VanderLei, and S. Archer: OSGi and Equinox: Creating Highly Modular Java Systems, Addison-Wesley, 2010

[31] Apache Community, Felix – OSGi framework, last accessed: June 2010, http://felix.apache.org/site/index.html

[32] MakeWave, Knopflerfish OSGi, last accessed: June 2010, http://www.knopflerfish.org/

[33] ProSyst, mBS Mobile SDK, last accessed: June 2010, http://www.prosyst.com, http://www.prosyst.com/index.php/de/html/content/64/mBS-Mobile-SDK/

[34] Apache Community, ServiceMix 4.2, last accessed: June 2010, http://servicemix.apache.org/2010/04/27/servicemix-420-released.html, http://servicemix.apache.org/home.html

 ISBN: 978-1-61208-105-2