Dynamic Software Updates
Programming Languages Research Seminar(MTAT.03.271)

Tartu 2012
# Table of Contents

ABSTRACT .................................................................................................................................. 3  
1. INTRODUCTION .................................................................................................................... 4  
2. DSU PROPERTIES, REQUIREMENTS AND GOALD .............................................................. 5  
2.1 EXISTING SOLUTIONS ........................................................................................................ 5  
3 CONCEPT .................................................................................................................................. 6  
   3.1 System DSU ........................................................................................................................ 6  
   3.2 Software DSU .................................................................................................................... 6  
      State transfer ........................................................................................................................ 6  
      Dynamic Linking .................................................................................................................. 7  
      Re-linking ............................................................................................................................ 7  
4. TECHNIQUES .......................................................................................................................... 8  
   4.1 Binary code rewriting ........................................................................................................ 8  
      Standard approach ............................................................................................................... 8  
      Binary Rewriting in Virtual machines .................................................................................. 8  
   4.2 Proxies objects ................................................................................................................ 8  
   4.3 State functions ............................................................................................................... 9  
   4.4 'Good' moment approach ............................................................................................. 9  
      Safe point ............................................................................................................................ 9  
      Stack search ........................................................................................................................ 9  
      Pause .................................................................................................................................... 9  
   4.5 Code Static Analysis ....................................................................................................... 10  
   4.6 Replication .................................................................................................................... 10  
   4.7 Update transactions, Rollbacks ..................................................................................... 11  
5. CONCLUSION ....................................................................................................................... 12  
REFERENCES ............................................................................................................................ 13
ABSTRACT

The usual approach to a software upgrade involves the developer providing a new executable, the client halting execution of the existing system and restarting with the new. In some cases restart can take several minutes to be validated and performed even if the fix may simply consist in replacing one method implementation without any side effect, like changing method signature. Dynamic Software Updating (DSU) addresses this difficulty by permitting programs to be updated while they run. This might be important for critical application, because many of them cannot afford to stop service. Examples range from pacemakers to cell phone base stations and nuclear power plant or credit card providers. Even such critical applications must be updated to fix bugs and add new features.

The main advantage of DSU is its ability to preserve program state during an update. For programs what are relatively stateless rolling stop-and-restart upgrades can exploit redundancy to permit updates with little disruption. However, many programs are a poor match for rolling-upgrade–based techniques because the running instances maintain critical state. DSU allows those active connections to immediately benefit from important program updates, whereas rolling upgrades would not.

This paper is a review of existing concepts and techniques for implementation of Dynamic Software Updating (DSU) problem solution.
1. INTRODUCTION

Unfortunately, DSU research community not reached a consensus on standard definition and classification of different types of DSU. But in this review i will consider types identified by Grupta[1]

1) Changes in structure of the system

2) Changes in execution code

In the beginning I will specify common requirements for DSU implementation. There's two well known ways to approach DSU – system approach and software approach. Since system approach is more about use of redundant hardware, where you have more than 1 machines available, and update them in turn. I will concentrate more on software approach, where main focus is transformation of the old process state into new process state. The main challenge is to discover mechanisms that allow to build dynamic changes to a system. Dynamic update usually contains three steps – Dynamic linking, Re-linking and State transfer.

Nowadays virtual machines became very popular, languages like Java or .Net is used by most of modern enterprise solution. Implementation of DSU for virtual machines in general is similar, but because Java byte code should be the same for many platforms it makes task easier, because there's no need to implement patches for certain architecture.

The main goal of this review is to analyze existing concept and techniques related with dynamic software updating and compare how they are implemented in different solutions
2. DSU PROPERTIES, REQUIREMENTS AND GOALD

There's variety of requirements and goals for DSU mechanism. In this section I list most common requirements that any DSU solution should ideally meet:

1. Avoidance of a restart.
   Essential concept in DSU. Application should run while updating process is in progress. This is the only 'must be' requirement, what determine is it dynamic software update or not.

2. Safety.
   DSU should provide safety guarantees. System should minimize the risk of errors after updates. Developers should always be interested in ensuring that the program will not crash by maintaining type safety. With a patch we not only have to check that it is safe to apply, but also that it is safe to apply with respect to a program state, or set of program states.

3. Efficiency
   The less complicated the updating process is, the less error-prone it will tend to be. Patches should be easy to write, and updates should not affect application performance

4. Flexibility
   Any part of a running system should be updateable without requiring downtime [2]

2.1 EXISTING SOLUTIONS

There's no 'silver bullet' general-purpose system that meets all of criteria s. Many systems have limited flexibility.

1. Ksplice
   Ksplice is an open source extension of the Linux kernel which allows system administrators to apply security patches to a running kernel without having to reboot the operating system. It uses a diff to determine changes between current and updated versions of Linux, and then uses binary rewriting to insert the changes into the running kernel. Ksplice is unique among DSU systems in that it supports a single target program (Linux kernel)

2. Ginseng
   DSU system to use the cons-freeness safety technique, allowing it to update functions that are live on the stack as long as they do not make concrete accesses to updated types.
3. UpStare

DSU system that uses a unique updating mechanism, stack reconstruction. To update a program with UpStare, a developer specifies a mapping between any possible stack frames.

4. Erlang

Programming language designed to support dynamic software updates, but does not guarantee safety, instead for developers is recommended to use defensive programming style.

3 CONCEPT

Most of the techniques used in DSU can be divided into two groups – system or software DSU.

3.1 System DSU

Idea is very simple, lets imagine we have two machines A and B. One of them runs application, and than when upgrade is required, B start to run new application and then state from machine A is transformed to machine B. The main challenge is state transformation. Some data like open stream, file handlers is very hard to transfer on the fly. This solution is fine, if no state transfer is needed, otherwise you need to implement data transformation.

3.2 Software DSU

The main complexity is to implement mechanisms that allow us to structure dynamic changes to a system. We can divide this task into three parts: dynamic linking, re-linking and state transfer.

State transfer

A function which transforms a state object or group of objects is referred to as a transformer function or state transformer. We face such problems, like storage expansion, where locations in memory for integers are upgraded to doubles or vice versa. Or in some case we need to upgrade whole data structure, like if we had singly-linked list, may now be required to be doubly-linked, we need to update every element in the list.
**Dynamic Linking**

Well known approach, used not only in case of dynamic software update. New code and data are designed to be loaded from shared libraries. We could say that dynamic linking is the opposite of state transfer. Instead of move a programs' state to a different one, we move the program to the state. Dynamic linking is popular and easy to implement. But this approach have many disadvantages in flexibility, because most common use is when we implement plug-in design. In that case only parts of the system(plugin itself) can be updated. A prototypical implementation is dlopen which can load data and code from shared libraries into address space of process and provides a mechanism to locate items in the library be name.

![Dynamic Linking Diagram](image)

*Figure 1.1: Plug-in extensibility: extensions are "plugged-in" to an extension interface in the running program.*

**Re-linking**

After new library is loaded, we need to ensure that old code have access to new library. Existing binders in the old code are re-linked, to the new one's. With relinking, the process of updating is active: the dynamic linker must go through the entirety of the program and ‘fix up’ any existing code to point to the new code. With reference indirection, updating is passive: the existing code is compiled to notice changes. As a result, the linker does not need to keep track of the existing code and simply makes changes to the table, but at the cost of an extra indirection to access definitions through the table. In both cases it is the responsibility of the state transformer function to find references to old definitions that are stored in the program’s data. For example, if the program defines a table of function pointers, the state transformer must redirect each pointer in the table to its new version.
4. TECHNIQUES

There's many technical way's to solve DSU. In this section we describe the most know approaches. It's good to know in what situation what technology is better.

4.1 Binary code rewriting

Binary rewriting is the process of writing low-level code into the memory image of a running native program to re-direct functions. While this requires no static analysis of a program, it is highly platform-dependent

Standard approach

Modifications are applied at a binary level, what means you modify binary code of executable file and code laded into memory. This technique have many disadvantages. Most of them comes from low-level nature of this approach. It depends on many things like – compiler, hardware architecture. Developers need to understand very level of the exact machine language used by processor. This technique is difficult to automate, because every update depends on original and new version of program.

Rewriting itself should be done very careful, before updating the code of function, we need to be sure it is not currently executed.

Tools like diff is used to compute difference between two versions (patch). In addition to patch, the programmer can define transformation function.

Binary Rewriting in Virtual machines

Using this technique for virtual machines like jvm is very similar, but because Java bytecode should be the same for many platforms it makes task easier.

Virtual machine specific problem in Dynamic updating is the version barrier. You need to create new classloaders and use them load new java classes. Problem is the the new classes are not easily accessible from classes loaded by different classloaders. To solve this dynamic proxies are used. The act like intermediaries between real classes and client classes.

Many existing tools offer run-time bytecode manipulation (CGLIB[5], Jrebel[4], Javeleon[6])

4.2 Proxies objects

This solution means, we need to add additional proxy layer to our application architecture. For example if we have client-server architecture, we could instead of having the client call
service methods, it calls some proxy, that points to current implementation of service. This approach is briefly reviewed by Miedes [3]

**4.3 State functions**

Idea is very simple every updatable part of our application need to implement two methods get_state() and set_state(state). And with every patch we need to implement transformer function update_state, who takes old state of object converts data to new state and calls setter methods.

The main disadvantage of this approach that programmer should by provided by programmer, what makes whole process very complex.

**4.4 'Good' moment approach.**

Before updating the component, our updater should somehow know that the update happens in right moment. It means process try to ensure that object is in stable state. To determine when is a good moment to do update several techniques are used:

**Safe point**

Idea is that we force our application to enter some dummy function, just to be sure that update can be applied safely. Of course for program kind of freezes for that updating time, and we still need to find good moment to enter such function. Other approach, is to directly point in code when is a good moment to update this particular data structure.

**Stack search**

Analyze stack of a process to find if a certain function of program is currently executed. If there's no address of function in stack it's safe to dynamically update this function.

**Pause**

Very similar to safe point approach, the only difference that we need to have “update manager” that decides when to pause our application to make updates. And also many application may became broken after resuming, so this adds complexion for developers, to make code pause-safe.
4.5 Code Static Analysis

To detect in which points of the programs is safe to perform dynamic update or to find where dynamic update should not be performed at all we could analyze code. The main idea around this analysis is to protect the state of program, so updates do not change state to inconsistent one. For example we could do update in the very beginning of the function or if function is read only. Another way to do analysis is to compare source code of new version with current version to build list of patches, so we could apply patch dynamically.

4.6 Replication

The idea is not actually apply update dynamically, but install the new version of the software in new replicas and shut down old ones. And then to promote to main replica on of new replicas.

Wang [11] proposes to dynamically change the consistency of a replicated system. He thinks that the consistency needs of a distributed system needs to be changed in run-time, depends on rates of read and write operations issued by the clients. The rate of read an write operations may be low or high, that means at any moment, the system can be in any of the four possible combinations.

The first combination is the read rate is high and the write rate is low. Write operations sent to any replica are redirected to the master, which redirects them to all the replicas, thus achieving full consistency among all the replicas. Read requests are sent to any replica and can attended immediately because they are all updated.

The second combination is both the read and write rates are high. Write operations sent to any replica are redirected to the master. The master forwards them to the deputy nodes. Read requests are sent to any replica and attended immediately.

The third combination is read rate is low and the write rate is high. Write operations sent to any replica are resent to the master, and then to all the deputy nodes. Read requests sent to the master or to any of the deputy nodes are responded immediately. Read requests

The fourth combination is read and write rates are low. The only different case is when a child node receives a read request. If the child it too outdated, it retrieves the requested value from the closest deputy node and returns it to the client.
4.7 Update transactions, Rollbacks

Some DSU implementation may offer the undo an update for instance when it is not behaving correctly or for any other reason. For this, it is enough to apply reverse patch. Brown and Patterson[12] propose a model for rollback mechanisms, as a solution to the external inconsistency problem. This happens when the rollback of an update made to an application also discards changes to the data that have been seen by the user.

The proposed model is based on three stages or steps: rewind, repair and replay. In the rewind step, the rollback mechanism rollbacks the changes to data made after applying the update. Previously, the rollback mechanism saves a semantic representation of those changes, so they can be re-applied later. In the repair step, the update is rollbacked. In the replay step, the saved changes are re-applied, over the rollbacked version of the application.
5. CONCLUSION

This paper review existing concept and techniques related with dynamic software updating and compare how they are implemented in different solutions. First, we specify common requirements and properties of DSU system. Then we analyze what methods help us to solve this problem. Mainly DSU solution is divided into two groups- system approach and software approach. Most popular concepts and techniques of software approach were introduced with reference to real implementation.
REFERENCES

[1] – Grupta, Runtime Updates

[2] -MICHAEL HICKS, Dynamic Software Updating,

[3] - Emili Miedes and Francesc D. Dynamic Software Update


[7] Iulian Gheorghe Neamtiu Practical Dynamic Software Updating


