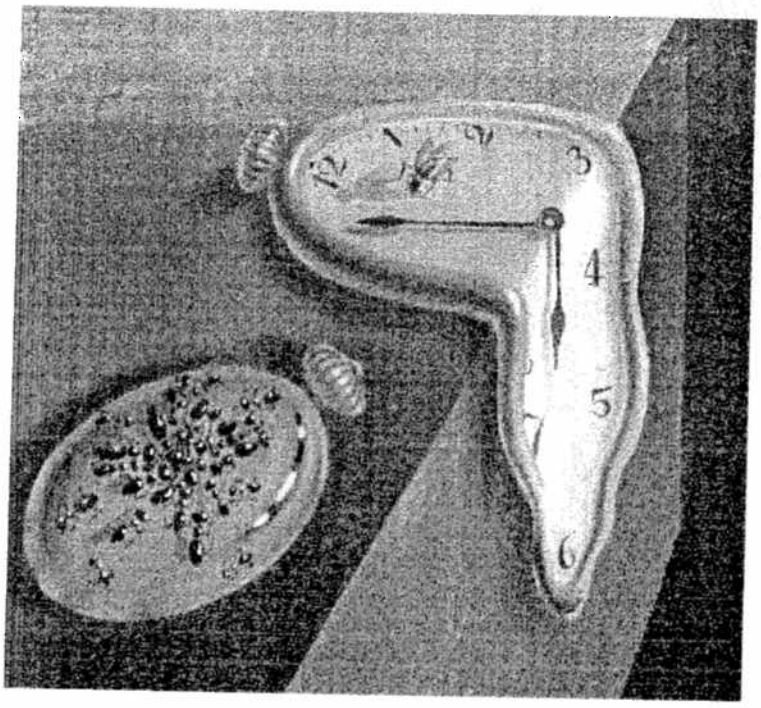
Institut für Automation
Abt. für Automatisierungssysteme

Technische
Universität
Wien

Projektbericht Nr. 183/1–13
November 1990

Real–Time Simulation (RTS) – Beschreibung

S. Stöckler



Ausschnitt aus: Salvador Dali, "Die Beständigkeit der Erinnerung"

RTS

Beschreibung

Stefan Stöckler
November 1990

1 Einführung

Der *Real Time Simulator* für (diskrete) technische Prozesse (im folgenden kurz *RTS* genannt) stellt ein System dar, mit dessen Hilfe die verschiedensten digitalen Prozeßleitsysteme auf ihre Funktionalität und Sicherheit getestet werden können. Der RTS ermöglicht, technische Prozesse im Labor zu simulieren, sodaß die meist auf Mikroprozessorbasis aufgebauten Prozeßsteuereinrichtungen kostengünstig und ohne Gefährdung der Umwelt überprüft werden können.

Im folgenden werden nun die Anwendungsgebiete und der allgemeine Aufbau des RTS ausgeführt.

1.1 Anwendungsgebiete

Der RTS kann prinzipiell überall dort zur Anwendung kommen, wo Prozeßleitsysteme entwickelt werden, bei denen es (aus welchen Gründen auch immer) schwierig ist, den technischen Prozeß, der beeinflußt werden soll, zu Testzwecken heranzuziehen.

Dies ist zum Beispiel dann der Fall, wenn durch einen Fehler im zu überprüfenden Leitsystem eine Gefährdung für Menschen, Tiere oder Umwelt entstehen könnte (z.B. Steuer- und Regeleinrichtungen für moderne Flugzeuge), oder aber der technische Prozeß selbst gar nicht zur Verfügung steht, wenn etwa Mikroprozessorsteuerungen für den Weltraumeinsatz entwickelt werden. In vielen Fällen wird es auch zu kostspielig sein, die entsprechenden Anlagen (Chemiewerke, Fertigungsstraßen in computergesteuerten Fabriken, Kraftwerke, ...) für die Entwicklung spezieller Steuersysteme abzustellen.

In allen oben genannten Fällen erscheint es also sinnvoll, einen frei konfigurierbaren Simulator zur Verfügung haben, der es ermöglicht, den technischen Prozeß so nachzuvollziehen, daß die entsprechenden Tests im Entwicklungslabor durchgeführt werden können. Darin ist im übrigen auch ein betriebswirtschaftlicher Aspekt, der für den Einsatz des RTS spricht, zu sehen, denn mit diesem Hilfsmittel kann die gesamte Entwicklung bis hin zu den Endprüfungen im Labor geschehen, was eine Verlagerung des Systems zu Testzwecken unnötig macht.

Um einen technischen Prozeß mit dem RTS simulieren zu können, muß dieser zuerst auf *Signalebene* definiert werden, wozu vor allem graphische Elemente (auf einer sogenannten *graphische Benutzeroberfläche*) verwendet werden. Ist ein Prozeß vollständig und konsistent definiert, so kann er auf einem Mikroprozessorsystem mit der entsprechenden *Prozeßperipherie* in *Echtzeit* simuliert werden.

Der RTS ist jedoch kein System, das nur deterministische Signale erzeugt, sondern die möglichen Fehlerfälle, die bei der Spezifikation eingearbeitet werden müssen, werden bei der Simulation berücksichtigt, d.h. die im Normalbetrieb zu erwartenden Störfälle treten mit den entsprechenden Wahrscheinlichkeiten auch während der Simulation auf. Außerdem können auf Wunsch auch spezielle Fehlerfälle während der Testphase direkt ausgelöst werden, um das Leitsystem auch in *Streßsituationen* zu prüfen.

2 Aufbau und Funktionsweise

Entsprechend den im vorigen Kapitel erwähnten Teilaufgaben (Definition, Simulation und direktes Eingreifen in die Simulation eines technischen Prozesses) gliedert sich der RTS in die drei logische Einheiten *Design Station* (*DS*), *Simulation Station* (*SS*) und *Interaction Station* (*IS*). Dabei können die Aufgabengebiete dieser Einheiten etwa folgendermaßen umschrieben werden:

1. Die Design Station (DS) stellt die graphische Oberfläche zur Spezifikation eines technische Prozesses zur Verfügung. Mit Hilfe dieses Werkzeuges (Tools) wird der Ablauf des Prozesses im Normal- wie auch in allen möglichen Störfällen spezifiziert.

   Ist die Definitionsphase abgeschlossen, so überprüft die DS die Angaben auf ihre Konsistenz und Durchführbarkeit. Letzteres begründet sich darin, daß zum Beispiel für eine gewünschte Anzahl von Signaltypen auch die entsprechende Menge von Ausgängen (bzw. Eingängen) im RTS vorhanden sein müssen. Vor allem aber muß überprüft werden, ob mit der vorhandenen Rechnerkapazität das definierte System überhaupt in Echtzeit simuliert werden kann, oder ob zusätzliche Mikroprozessoren notwendig wären.

   Hat eine *Prozeßspezifikation* die oben genannten Tests erfolgreich passiert, so wird sie automatisch in entsprechende Programme (*Simulator–Software*) übersetzt, die auf der *Simulator–Hardware* ablauffähig sind.

2. Die so erzeugte Software kann nun mit Hilfe der Simulation Station (SS) in die Simulator–Hardware geladen und gestartet werden. Die Hauptaufgabe der SS ist es, die Simulator–Software auszuführen und dadurch den auf der DS spezifizierten technischen Prozeß an den Ein- und Ausgängen der Prozeßperipherie zu simulieren, dabei arbeitet sie sehr eng mit der IS zusammen, die unter Punkt 3 näher beschrieben wird.

3. Auf der Interaction Station (IS) werden laufend Prozeßstatistiken erstellt und ausgegeben, so daß es zu jedem Zeitpunkt möglich ist, sich ein Bild vom Zustand des Gesamtsystems zu machen.

   Ein besonders wichtiger Aspekt der IS ist aber, daß sie die Möglichkeit zum direkten Eingriff in das Prozeßgeschehen bietet. Dies kann zum Beispiel nötig werden, um für Testzwecke bewußt Streßsituationen zu erzeugen oder bestimmte Ein- oder Ausgänge als deffekt zu behandeln.

   Über die IS ist also jederzeit ein direktes Eingreifen in die Simulation möglich, wobei die entsprechenden Rückmeldungen über den Prozeßstatus ebenfalls sofort angezeigt werden.

Neben der logischen Gliederung liegt beim RTS auch eine technisch bedingte vor. Während die DS vollständig auf einer Workstation implementiert ist, befinden sich die IS teilweise und die SS fast zur Gänze auf dem *Mikroprozessorsystem*. Die Teilsysteme sind über ein Netzwerk miteinander verbunden, so daß ein ungehinderter Datenaustausch möglich ist. Das Simulations–Hardwaresystem setzt sich aus einem oder mehreren *Mikroprozessor–Boards*, die über das Netzwerk verbunden sind und den zugehörigen Ein-/Ausgabe–Boards, zusammen.

Die SS, die auf die verschiedenen Mikroprozessorsystemen verteilt abläuft, erzeugt entsprechend der Prozeßspezifikation alle Signale des technischen Prozesses. Die Steuereingriffe des Prozeßleitsystems werden von der SS verarbeitet und beeinflussen in entsprechender Weise die Simulation.

Das folgende Bild zeigt den schematische Aufbau des RTS, wobei versucht wird sowohl die technischen als auch die logischen Komponenten darzustellen.

1

Bild 1: Aufbau des RTS

3 Design Station

Auf der Design Station (DS) wird vom Anwender des RTS der technische Prozeß, der simuliert werden soll, mit Hilfe einer graphischen Benutzeroberfläche spezifiziert. Darum muß die DS auf einer Workstation mit graphikfähigen Bildschirm und Maus installiert werden.

Da die DS nicht nur für die Spezifikation des Prozeßmodelles, sondern auch für dessen Verifikation zuständig ist, gliedert sie sich wiederum in mehrere logische Einheiten: *Specification Tool*, *Verification Tool* und *Code Generation Tool.*

3.1 Specification Tool

Mit Hilfe des Specification Tools (DS-ST) definiert der Benutzer das Verhalten des diskreten technischen Prozesses (kontinuierliche Vorgänge müssen über entsprechende Funktionen angenähert werden). Dazu müssen zum einen Art und Anzahl der Ein- und Ausgänge und zum anderen die inneren Zusammenhänge des Prozeßablaufes spezifiziert werden. Ersteres ist nötig, da der Simulator verwendet wird, um ein bestehendes Prozeßleitsystem zu testen, und zu diesem Zwecke an dieses angeschlossen werden muß, wobei die erforderlichen Eigenschaften der Signale (analog oder digital, Wertebereich, ...) durch die Simulation erzeugt werden müssen.

Da komplexe technische Prozesse im allgemeinen in parallel ablaufende, einfachere Teilprozesse zerlegt werden können, spezifiziert der Anwender nach dem Bottom Up-Verfahren zuerst einfacheste Systemzusammenhänge, um durch Verknüpfung derselben in weiteren Arbeitsschritten die endgültige Struktur des zu spezifizierenden Prozesses zu modellieren.

Dabei stehen dem Benutzer als Arbeitserleichterung häufig benötigte Systemelemente in einer Bibliothek zur Verfügung, sodaß eine wiederholte Neudefinition dieser Komponenten entfällt. Außerdem ist es dem Anwender möglich komplexere oder spezifische, aber immer wieder auftretende Elemente in eine eigene Bibliothek aufzunehemen, um sie so für weitere Simulationen zu erhalten und zugänglich zu machen.

Wie oben schon erwähnt handelt es sich in den meisten Fällen um parallele Abläufe, die simuliert und somit auch zuvor spezifiziert werden müssen. Das Simulation Tool bietet hiefür dem Benutzer eine graphische Oberfläche, die diese Eigenschaft optisch leicht erfassbar darstellt und somit das Formulieren solcher Systeme unterstützt.

Die Spezifikationsmethode beruht auf der Annahme, daß für jeden Prozeß (jedes technische System) eine endliche Menge an Zuständen gefunden werden kann, wobei sich das System zu jedem Zeitpunkt in einem bestimmten davon befindet. Wechselt es von einem Zustand in einen anderen, so findet dieser Übergang in unendlich kurzer Zeit statt.

Der Anwender spezifiziert nun die verschiedenen Zustände, in denen sich der Prozeß befinden kann, und die Bedingungen, die einen Zustandswechsel ermöglichen bzw. erzwingen. Jeder Zustand repräsentiert genau eine Wertekombination aller Systemgrößen (den Zustandsvektor). Wobei unter Systemgröße sowohl die Ein- und Ausgänge als auch innere Variablen des Prozesses verstanden werden.

3.2 Verification Tool

Nach erfolgter Spezifikation soll mit Hilfe des Verification Tools (DS-VT) festgestellt werden, ob die gemachten Angaben in sich konsistent sind und ob sie mit der zur Verfügung stehenden Simulator-Hardware, die in der Hardware Support Table aufgelistet ist, ausgeführt werden können.

Die erste Aufgabe des Verification Tools wird es sein, die statische Konsistenz der Spezifikation zu überprüfen. Darunter fallen Tests wie etwa das Überprüfen, ob überhaupt alle spezifizierten Zustände erreicht werden können, oder die Untersuchung, ob die definierten Ein- und Ausgabekanäle in der vorhandenen Simulator-Hardware zur Verfügung stehen. Die Informationen zu letzterem können der Hardware Support Table (HST), in der alle verfügbaren Komponenten vom Benutzer eingetragen sein müssen, entnommen werden.
Des weiteren sollte es über das in der Spezifikation definierte Zeitverhalten des Prozesses möglich sein, dynamische Konsistenzprüfungen zu machen (zum Beispiel, ob durch die Spezifikation gefordert wird, daß zu einem bestimmten Zeitpunkt ein und dem selben Ausgang verschiedene Werte zugewiesen werden sollen).

Aus dem obigen Schritt sollten außerdem Angaben resultieren, die es ermöglichen, die erforderliche Performance (Anzahl der Prozessoren), zu bestimmen und diese mit der vorhandenen Rechnerleistung (Hardware Support Table) zu vergleichen.

Wenn das Verification Tool Fehler in der Spezifikation findet oder die gemachte Spezifikation aus Gründen des zu geringen Hardware Supports nicht ausführbar ist, dann wird dies dem Anwender in entsprechender Weise mitgeteilt, so daß dieser eine Redesign des spezifizierten Prozesses vornehmen bzw. die noch benötigen Hardware-Komponenten beschaffen kann.

3.3 Code Generation Tool

Sollte die Verifikation positiv abgeschlossen worden sein, so ist es Aufgabe des Code Generation Tools (DS-CGT), die vom Verification Tool zur Verfügung gestellten Angaben über die erforderliche Performance zu nutzen und darauf aufbauend eine Aufteilung der einzelnen Simulations-Komponenten auf die vorhandenen Prozessoren zu machen. In diesem Schritt wird die sogenannte Hardware Configuration Table (HCT) erstellt, die dem Benutzer angibt, wie dieser die Simulator-Hardware aus den verschiedenen Elementen (Prozessor- und Ein-/Ausgabe-Boards) aufbauen muß.
Ist die Aufteilung der Rechnerleistung erfolgreich beendet, so generiert dieses Tool aus der Spezifikation und den Angaben aus der Hardware Configuration Table den Source-Code für die Simulator-Software. Dieser Source-Code wird dann in weiterer Folge automatisch compiliert bzw. assembliert, sodaß das Endprodukt des Code Generation Tools ein ausführbarer Object-Code für die einzelnen Prozessoren ist.

4 Simulation Station

Die von der Design Station, oder genauer gesagt vom Code Generation Tool, gelieferten Object-Modules werden mit Hilfe der Simulation Station (SS) in die Speicher der einzelnen Prozessoren geladen (Down Load über das Netzwerk; siehe Skizze Seite 4) und synchron gestartet. Die Hauptaufgabe der SS ist es, die Simulator-Software auf den Mikroprozessorsystemen abzuarbeiten und die von der Interaction Station (seihe Kapitel 5) geforderten Prozeßdaten zn dieselbe zu liefern.

5 Interaction Station

Während der eigentlichen Simulation des technischen Prozesses durch die Simulation Station erfolgt, stellt die Interaction Station (IS) das Bindeglied zwischen Benutzer und Simulation dar.

Die IS stellt Funktionen zur Verfügung, über die das direkte Eingreifen in den Simulationablauf möglich ist. So können etwa zuvor in der Spezifikation festgelegte Fehlerfälle auf Wunsch ausgelöst werden, um das Prozeßleitsystem auch in Streßsituationen zu testen.

Aber auch während der unbeeinflußten Simulation dient die IS als Kommunikationskanal zwischen Simulation und Benutzer. So hat letzterer die Möglichkeit zur Definition von Prozeßdaten, die während der Laufzeit über die Interaction Station dynamisch angezeigt werden sollen. Um eine sinnvolle Darstellung zu ermöglichen bietet die IS außerdem eine Auswahl von Ausgabevarianten (als einfaches Display (Ziffernausgabe), als Liste (Ziffernausgabe, untereinander), Balkendiagramm , Tortendiagramm, ...).

Komplexere Statusmeldungen können aus der Verknüpfung einzelner Prozeßdaten gewonnen und dargestellt werden.

Durch diese dynamischen Prozeßstatusausgaben, die während der Laufzeit neu- oder umdefiniert werden können, hat der Benutzer den gewünschten Überblick über den Simulationsvorgang.

Neben den dynamischen Ausgaben stellt die IS auch die Möglichkeit statistischen Auswertung bestimmter Prozeßdaten (wie Mittelwert, Varianz oder Korrelationen) zur Verfügung. Diese Auswertungen können ebenso während der Laufzeit definiert und gestartet bzw. gestoppt werden.

6 Problemstellungen

In diesem Kapitel folgt eine Aufstellung der zu lösenden Probleme, wobei diese nur grob bezüglich der Strukturierung durch DS, SS und IS geordnet sind. Allgemein gilt, daß die (graphische) Benutzeroberfläche natürlich für alle drei Komponenten die selbe sein soll, sodaß für den Anwender die Aufgliederung in verschiedene Systemteile (DS, IS und auch SS) transparent gemacht wird.

6.1 Aufgaben bezüglich DS

Die Aufgaben die sich bezüglich der Design Station (DS) stellen, lassen sich noch einmal in die drei logischen Abschnitte Spezifikation, Verifikation und Code-Generierung einteilen (siehe auch Kapitel 3).

6.1.1 Spezifikation des technischen Prozesses

- Spezifikation verschiedenster Signale (Signalklassen → Klassifizierung der möglichen Signale durchführen (zyklische, digitale Signale, indeterministisch auftretende digitale Signale, definierte Impulsfolgen, analoge Signale mit stochastischen Verläufen oder mit einfachen zeitlichen Abhängigkeiten wie etwa Sinus- oder Exponentialfunktionen) und der Verbindungen der Ein- und Ausgänge mit den sie beschreibenden Zustandsvariablen.

- Definition von Ereignissen (Events), auf die reagiert werden kann (→ Zustandsübergänge) → einfache und zusammengesetzte (aus einfachen) Ereignisse.

- Spezifizierung von jederzeit möglichen Zustandsänderungen, die etwa durch bestimmte Vorgänge an einem oder mehreren Eingängen ausgelöst werden.
  Überlegung: bei Spezifikation mittels Petri-Netzen zusätslich Regeln der Art:
  `if Event_x then enter State_y execute Action_z`
  Das Auftreten des Ereignisses Event_x (das könnte ein Interrupt, die fallende Flanke eines bestimmten Signals oder das Zusammentreffen mehrerer einfacher Ereignisse sein) löst unabhängig vom Vorzustand einen Wechsel in den Zustand State_y und zusätzlich die Aktion Action_z (das wäre zum Beispiel das Setzen eines bestimmten Signalausganges auf 'HIGH' oder ähnliches) aus.

- Spezifikation stochastischer Abweichungen des Prozesses vom "Normalverhalten"

- Spezifikation der möglichen fehlerfälle, ihrer Auftrittswahrscheinlichkeiten und ihrer Korrelationen.

- Spezifikation analoger Signale.
  Lösungsmöglichkeit:

Definition als diskrete Signale, die mit entsprechend kleiner Zykluszeit (charkteristische Systemgröße, auf jeden Fall kleiner als die Abtastzeit des Prozeßleitsystems) auf einen aktuellen Stand gebracht werden.

- Definition einer (minimalen) Menge von Signalklassen und Zurverfügungstellung einer Bibliothek mit Generic-Objects für jede Klasse

- Möglichkeit für den Benutzer eine eigene Bibliothek (zusammengesetzer) spezieller Signal-Objekte aufzubauen.

- Möglichkeit für den Benutzer eine Bibliothek von Teilspezifikationen aufzubauen, wobei jede Teilspezifikation einmal oder öfter in einer übergeordneten Spezifikation verwendet (eingebunden) werden kann.

Als Basis für die Spezifikation des Prozeßverhaltens bieten sich adaptierte *Petri-Netze* an, da diese ursprünglich für die Darstellung paralleler, diskreter Vorgängen entwickelt wurden und schon entsprechende Theorien (Reachability Graph, stochastische Netze) dafür vorhanden sind.

6.1.2 Verifikation

- Übersetzung der Spezifikation in eine für automatische Verifkationswerkzeuge verständliche Darstellung (Logiksprache).

- Verifikation der statischen Konsistenz ("tote" Zweige in den Petri-Netzen, Boundedness, nicht entscheidbare Konfigurationen, ...).

- Verifikation der dynamischen Konsistenz (Zeitverhalten: z.B. "Besteht die Möglichkeit, daß einem Ausgang gleichzeitig zwei verschiedene Werte zugewiesen werden?" u.ä.).

- Bestimmung der nötigen Prozessor-Performance.

- Verifikation der Realisierbarkeit (vorhandene = geforderte Performance; Anzahl der spezifizierten Ein-/Ausgänge = Anzahl der vorhandenen, ...).

6.1.3 Code-Generierung

- Aufteilung der Last auf die (durch das Verification Tool (Performance)) geforderte Anzahl von Prozessoren. → Bestimmen der Hardware Konfiguration.

- Automatische Generierung der Simulator-Software für die einzelnen Prozessoren.

- Einbindung der nötigen Interprozesskommunikation in die Software (vor allem auch der Kommunikation über Prozessorgrenzen hinweg) → global konsistenter Systemzustand zu jedem Zeitpunkt.

6.2 Aufgaben bezüglich SS

- Download der Software in die lokalen Speicher aller Prozessoren und synchronisierter Start.
- Synchronisation der Uhren des verteilten Mikroprozessorssystems (unabhängig von der Anzahl der beteiligten Prozessoren) zur Aufrechterhaltung einer globalen Zeit, damit ein konsistenter Systemzustand gewährleistet werden kann.
- Sammeln der von der Interaction Station geforderten Prozeßdaten und weiterreichen derselben.
- Ausführung der über die Interaction Station getätigten Eingriffe in das Simulationsverhalten.

6.3 Aufgaben bezüglich IS

- Definition von einfachen und komplexen (zusammengesetzten) Prozeßdaten während des Simulationslaufes (→ Umdefinitionen möglich).
- Darstellung der definierten Prozeßdaten (aktueller Stand) in einer geeigneten (frei wählbaren) Form während der Simulation.
- Statistische Auswertung der Prozeßdaten.
- Möglichkeiten zum direkten Eingriff in das Simulationsgeschehen → Auslösen von (spezifizierten) Fehlerfällen.

7 Zeitplan/Personalplanung

7.1 Design Station

7.1.1 Specification Tool

1. Entwicklung von Methoden zur Spezifikation von (diskreten) technischen Prozessen (auf Signal-Level).
   Klaus Schoßmaier 11.90 bis 6.91 **Diplomarbeit.**

2. Entwicklung der Benutzeroberfläche und Implementierung der obigen Methoden unter UNIX, XWindows und OSF/Motif.
   N.N ~ ab 3.91 **Praktika oder Diplomarbeit.**

7.1.2 Verification Tool

3. Entwicklung von Methoden zur Verifikation der Prozeßspezifikation bezüglich statischer und dynamischer Konsistenz.
   N.N. ~ ab 5.91 **Diplomarbeiten oder Dissertation.**

4. Implementierung der obigen Methoden.
   N.N ~ ab 10.91 **Praktika oder Diplomarbeit.**

5. Entwicklung einer Methode zur Bestimmung der nötigen Prozessor-Performance.
   N.N. ~ ab 8.91 **Praktika oder Diplomarbeit.**

6. Implementierung der obigen Methode.
   N.N. ~ ab 11.91 **Praktikum.**

7.1.3 Code Generation Tool

7. Entwicklung und Implementierung eines Tools zur automatischen Code-Generierung.
   N.N. ~ ab 2.92 **Praktika oder Diplomarbeit.**

7.2 Simulation Station

8. Bereitstellen einer Test-Hardware (VME-Bus mit 68030-Prozessoren, Ether-Net, Clocksynchronisation (← übernehmen von Projekt: VTA)).
   N.N. ~ ab 3.92 **Praktikum.**

7.3 Interaction Station

9. Entwicklung der notwendigen Tools für das Definieren und Auswerten der zu beobachtenden Prozeßdaten. Implementierung und Einbindung in die bestehende Benutzeroberfläche.
   N.N. ~ ab 1.92 **Praktikum.**

7.4 Bibliothekserstellung und Test

10. Erstellen von Bibliotheken mit häufig gebrauchten Grundelementen (Generic Objects) für die Spezifikation und Entwurf von Testfällen.
    N.N. ~ ab 6.92 **Praktikum.**

11. Testphase.

Inhaltsverzeichnis

Bildverzeichnis